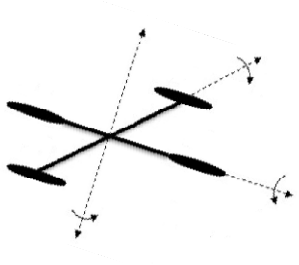
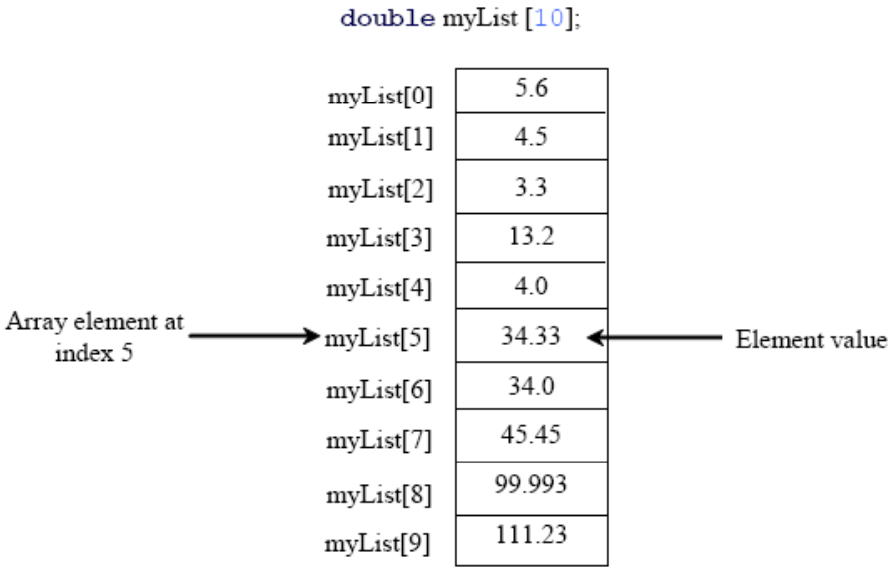
배 열

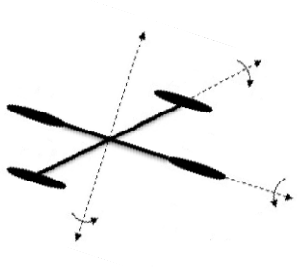


- 배열 (Array) : 같은 유형의 요소가 고정된 크기만큼 연속된 메모리 공간에 저장되는 자료 구조

```
datatype arrayRefVar[arraySize];
```

```
double myList[10];
```





- int size = 4;
- double myList[size]; // 잘못된 것

- const int SIZE = 4;
- double myList[SIZE]; // 잘된 것

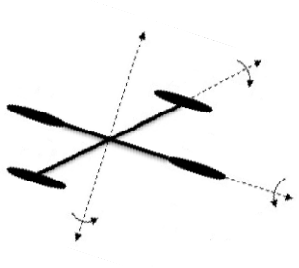
- 배열의 초기화

```
double myList[4] = {1.9, 2.9, 3.4, 3.5};
```

```
double myList[4];
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```

```
double myList[4];
myList = {1.9, 2.9, 3.4, 3.5};
```

→ 오류

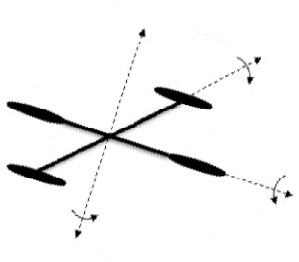


```
double myList[] = {1.9, 2.9, 3.4, 3.5};
```

초기화를 전체영역에서 수행할 때는 크기를 생략해도 됨.

```
double myList[4] = {1.9, 2.9};
```

일부만 초기화하는 것도 가능함.

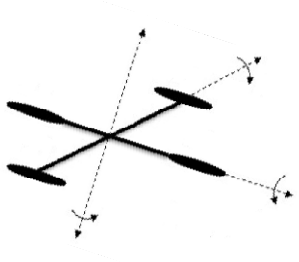


- 문자배열의 초기화

```
char city[] = {'D', 'a', 'l', 'l', 'a', 's'};
```

```
char city[] ="Dallas";
```

'D'	'a'	'l'	'l'	'a'	's'	'\0'
city[0]	city[1]	city[2]	city[3]	city[4]	city[5]	city[6]

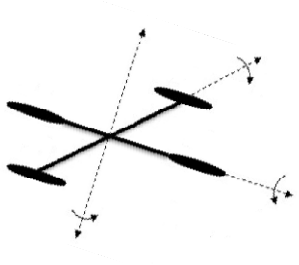


- 배열의 출력

```
for (int i = 0; i < ARRAY_SIZE; i++)
{
  cout << myList[i] << " ";
}
```

단, 문자 배열은 문장으로 출력 가능

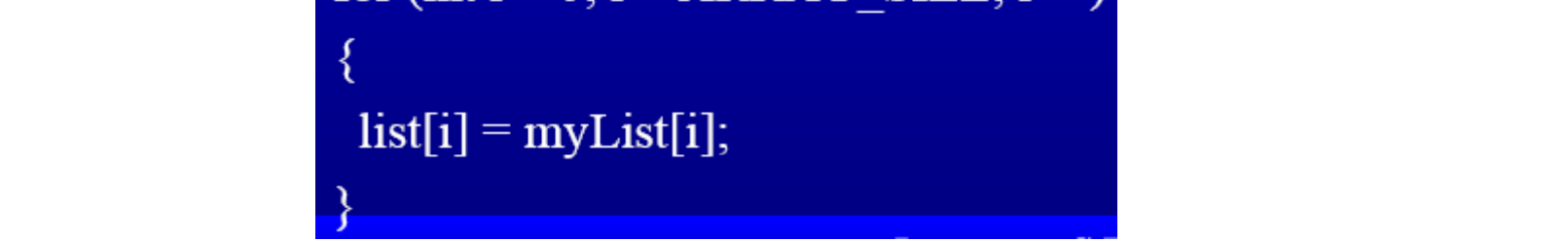
```
char city[] = "Dallas";
cout << city;
```



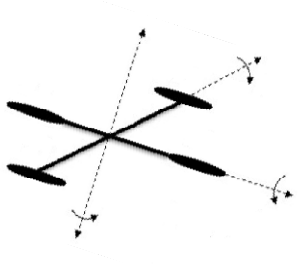
- 배열의 복사

```
list = myList;
```

→ 허용되지 않음

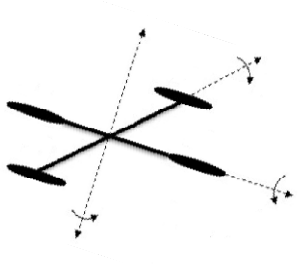


```
for (int i = 0; i < ARRAY_SIZE; i++)
{
  list[i] = myList[i];
}
```



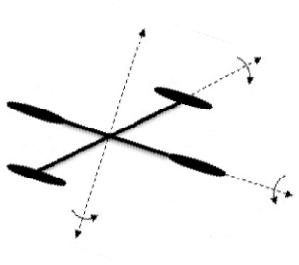
- 배열의 모든 요소의 합

```
double total = 0;
for (int i = 0; i < ARRAY_SIZE; i++)
{
  total += myList[i];
}
```



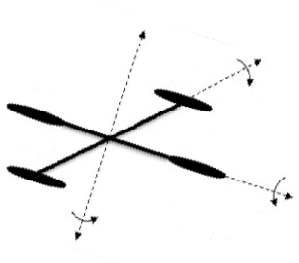
- 배열의 최대값 구하기

```
double max = myList[0];
for (int i = 1; i < ARRAY_SIZE; i++)
{
  if (myList[i] > max) max = myList[i];
}
```



- 배열의 최대값과 그 인덱스 구하기

```
double max = myList[0];
int indexOfMax = 0;
for (int i = 1; i < ARRAY_SIZE; i++)
{
  if (myList[i] > max)
  {
    max = myList[i];
    indexOfMax = i;
  }
}
```

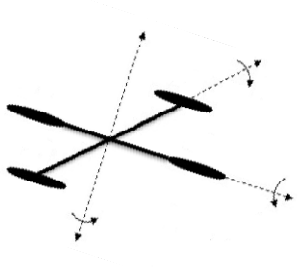


배열 변수 values를 선언하고 생성한 다음
배열의 레퍼런스를 values에 기억시킨다.

```
int main()
{
   int values[5];
   for (int i = 1; i < 5; i++)
   {
     values[i] = values[i] + values[i-1];
   }
   values[0] = values[1] + values[4];
}
```

After the array is created

0	0
1	0
2	0
3	0
4	0

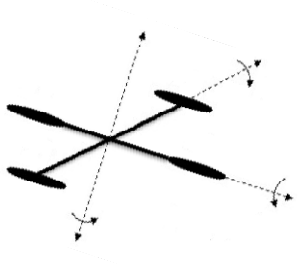


i는 1

```
int main()
{
   int values[5];
   for (int i = 1; i < 5; i++)
   {
     values[i] = values[i] + values[i-1];
   }
   values[0] = values[1] + values[4];
}
```

After the array is created

0	0
1	0
2	0
3	0
4	0

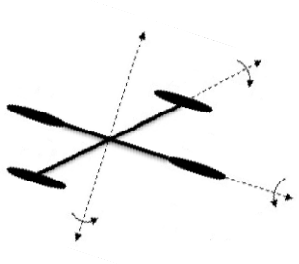


i (=1) 는 5 보다 작음

```
int main()
{
   int values[5];
   for (int i = 1; i < 5; i++)
   {
     values[i] = values[i] + values[i-1];
   }
   values[0] − values[1] + values[4];
}
```

After the array is created

0	0
1	0
2	0
3	0
4	0

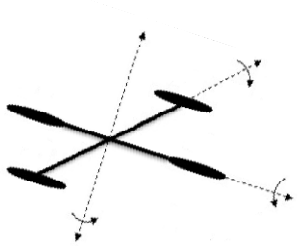


이 줄 수행한 다음 value[1] 은 1

```
int main()
{
    int values[5];
    for (int i = 1; i < 5; i++)
    {
      values[i] = values[i] + values[i-1];
    }
    values[0] − values[1] + values[4];
}
```

After the first iteration

0	0
1	1
2	0
3	0
4	0

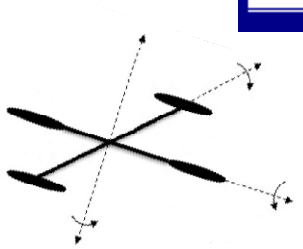


i++ 한 다음 i는 2가 됨

```
int main()
{
   int values[5];
   for (int i = 1; i < 5; i++)
   {
     values[i] = values[i] +
       values[i-1];
   }
   values[0] = values[1] +
      values[4];
}
```

After the first iteration

0	0
1	1
2	0
3	0
4	0

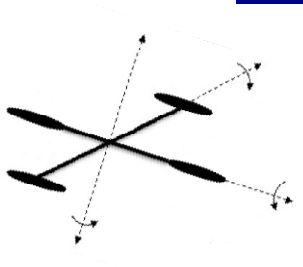


i (= 2) 는 5 보다 작음

```
int main()
{
   int values[5];
   for (int i = 1; i < 5; i++)
   {
     values[i] = values[i] +
       values[i-1];
   }
   values[0] = values[1] +
      values[4];
}
```

After the first iteration

0	0
1	1
2	0
3	0
4	0

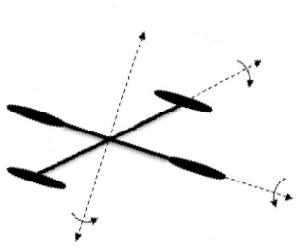


이 줄 수행한 다음 value[2] 는
3(2 + 1)이 됨

```
int main()
{
   int values[5];
   for (int i = 1; i < 5; i++)
   {
     values[i] = values[i] + values[i-1];
   }
   values[0] − values[1] + values[4];
}
```

After the second iteration

0	0
1	1
2	3
3	0
4	0

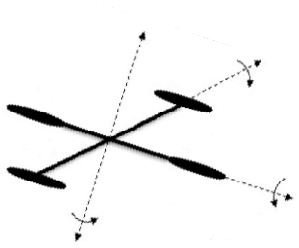


다음에 i는 3이 됨

```
int main()
{
   int values[5];
   for (int i = 1; i < 5; i++)
   {
     values[i] = values[i] + values[i-1];
   }
   values[0] − values[1] + values[4];
}
```

After the second iteration

0	0
1	1
2	3
3	0
4	0

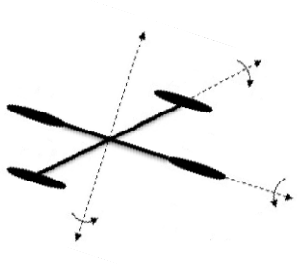


i (=3) 는 5보다 작음

```
int main()
{
   int values[5];
   for (int i = 1; i < 5; i++)
   {
     values[i] = values[i] + values[i-1];
   }
   values[0] = values[1] + values[4];
}
```

After the second iteration

0	0
1	1
2	3
3	0
4	0

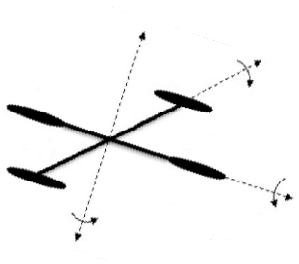


이 줄 수행한 다음 value[3] 은 6(3 + 3)이 됨

```
int main()
{
   int values[5];
   for (int i = 1; i < 5; i++)
   {
     values[i] = values[i] + values[i-1];
   }
   values[0] − values[1] + values[4];
}
```

After the third iteration

0	0
1	1
2	3
3	6
4	0

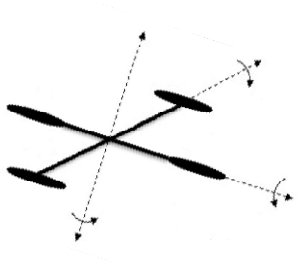


다음에 i는 4가 됨

```
int main()
{
   int values[5];
   for (int i = 1; i < 5; i++)
   {
     values[i] = values[i] + values[i-1];
   }
   values[0] − values[1] + values[4];
}
```

After the third iteration

0	0
1	1
2	3
3	6
4	0

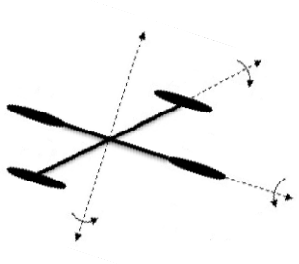


i (=4) 는 5보다 작음

```
int main()
{
   int values[5];
   for (int i = 1; i < 5; i++)
   {
     values[i] = values[i] + values[i-1];
   }
   values[0] − values[1] + values[4];
}
```

After the third iteration

0	0
1	1
2	3
3	6
4	0

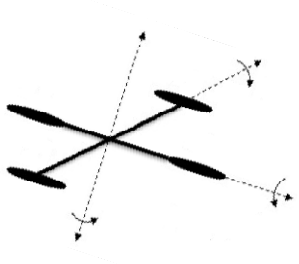


수행하면 value[4] 는 10(4 + 6)이 됨

```
int main()
{
   int values[5];
   for (int i = 1; i < 5; i++)
   {
     values[i] = values[i] + values[i-1];
   }
   values[0] − values[1] + values[4];
}
```

After the fourth iteration

0	0
1	1
2	3
3	6
4	10

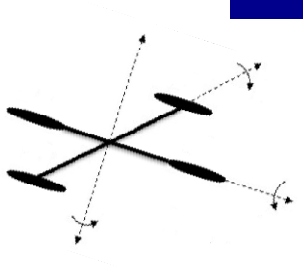


i++ 한 다음에는 i는 5가 됨

```
int main()
{
   int values[5];
   for (int i = 1; i < 5; i++)
   {
     values[i] = values[i] + values[i-1];
   }
   values[0] = values[1] + values[4];
}
```

After the fourth iteration

0	0
1	1
2	3
3	6
4	10

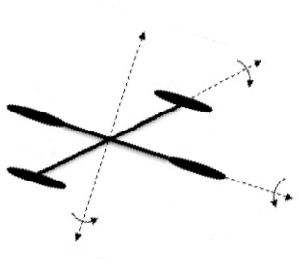


i (=5) < 5는 false 이므로 . 루프 끝

```
int main()
{
   int values[5];
   for (int i = 1; i < 5; i++)
   {
     values[i] = values[i] + values[i-1];
   }
   values[0] = values[1] + values[4];
}
```

After the fourth iteration

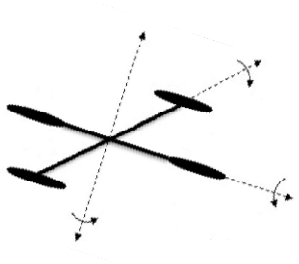
0	0
1	1
2	3
3	6
4	10



이 줄 다음에 values[0]는
11 (1 + 10)이 됨

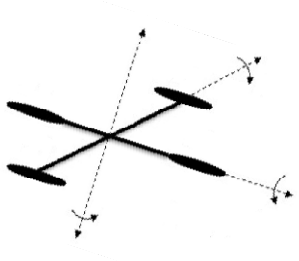
```
int main()
{
   int values[5];
   for (int i = 1; i <  ; ++)
   {
     values[i] =  alues[i] + values[i-1];
   }
   values[0] = values[1] + values[4];
}
```

0	11
1	1
2	3
3	6
4	10



- Reverse 함수

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```



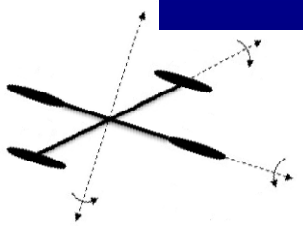
```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

i = 0, j = 5

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

list	1	2	3	4	5	6

newList	0	0	0	0	0	0

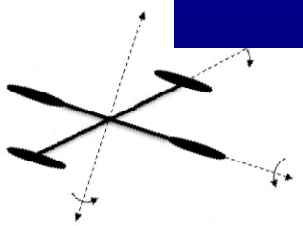


```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

i (= 0)는 6보다 작음

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

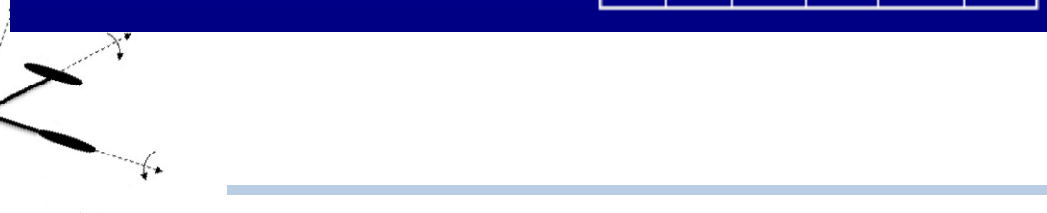
list	1	2	3	4	5	6
newList	0	0	0	0	0	0



```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

i = 0, j = 5
list[0]을 result[5]에 저장



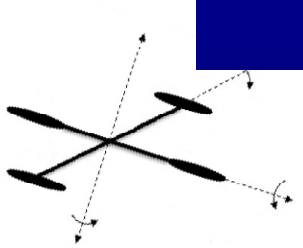
```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

수행 후에, i는 1,
j는 4가 됨

list	1	2	3	4	5	6

newList	0	0	0	0	0	1



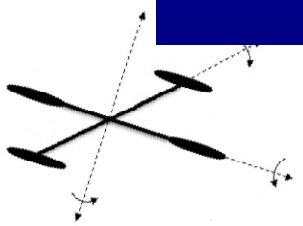
```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

i (=1)는 6 보다 작음

```
void reverse(const int list[], int newList[], int
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

list	1	2	3	4	5	6

newList	0	0	0	0	0	1



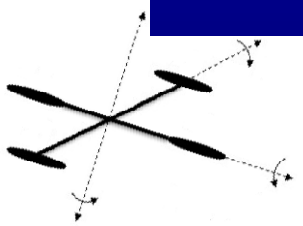
```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

i = 1, j = 4
list[1]를 result[4]에 저장

list	1	2	3	4	5	6

newList	0	0	0	0	2	1

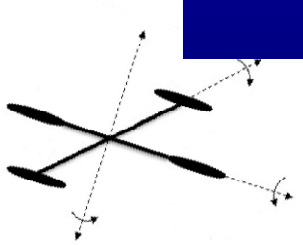


```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

수행 한 다음, i는 2, j는 3이 됨

list	1	2	3	4	5	6
newList	0	0	0	0	2	1

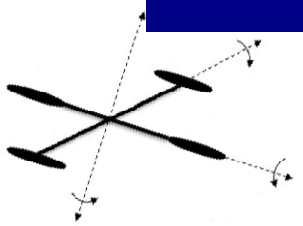


```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

i (=2)는 아직 6보다 작음

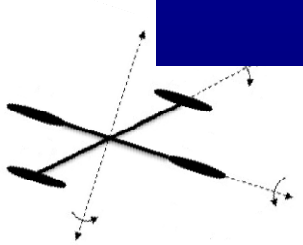
list	1	2	3	4	5	6
newList	0	0	0	0	2	1



```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

i = 2, j = 3
list[i]를 result[j]에 저장

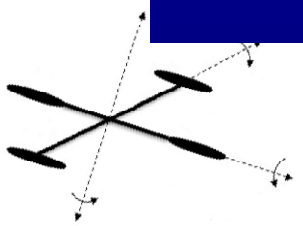


```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

수행 한 다음, i는 3,
j는 2가 됨

list	1	2	3	4	5	6
newList	0	0	0	3	2	1

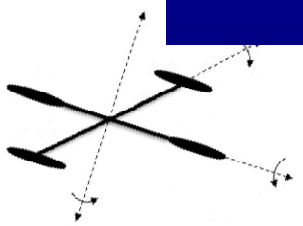


```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

i (=3)은 6보다 작음

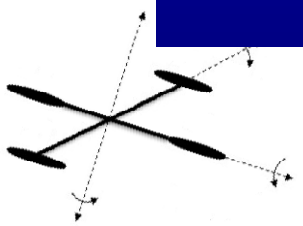
list	1	2	3	4	5	6
newList	0	0	0	3	2	1



```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

i = 3, j = 2
list[i]를 result[j]에 저장

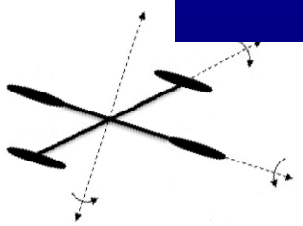


```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

수행 한 다음, i는 4,
j는 10이 됨

list	1	2	3	4	5	6
newList	0	0	4	3	2	1

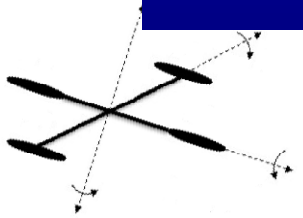


```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

i (=4)는 아직 6보다 작음

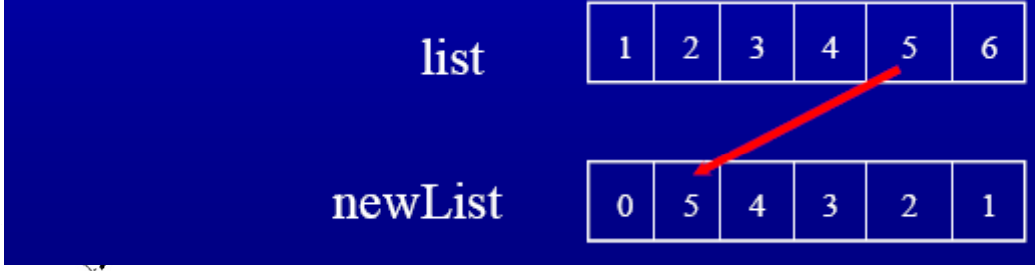
list	1	2	3	4	5	6
newList	0	0	4	3	2	1

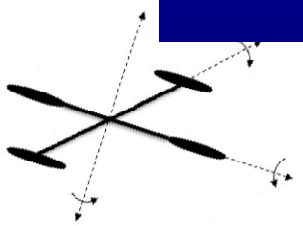


```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

i = 4, j = 1
list[i]를 result[j]에 저장





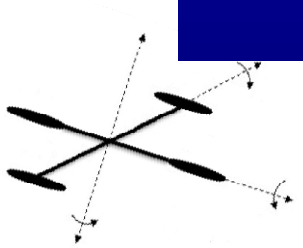
```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

수행 한 다음, i는 5,
j는 0이 됨

list	1	2	3	4	5	6

newList	0	5	4	3	2	1



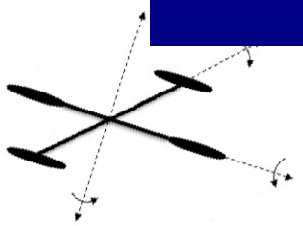
```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

i (=5)는 6보다 작음

list	1	2	3	4	5	6

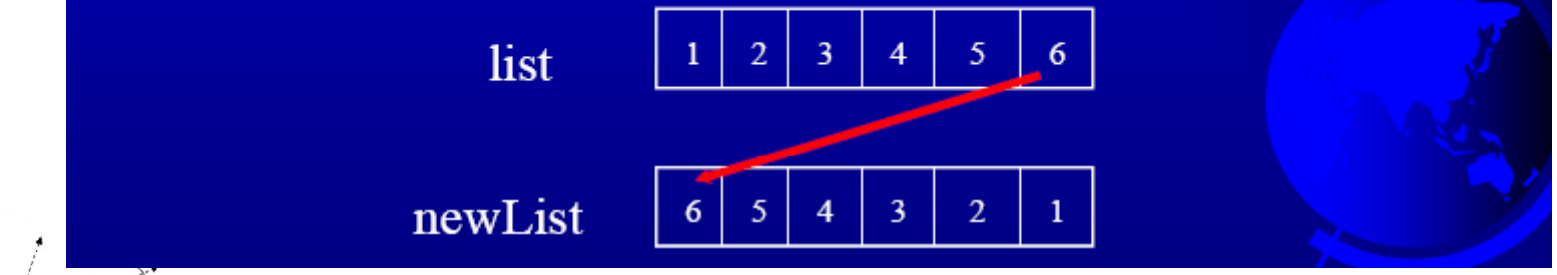
newList	0	5	4	3	2	1

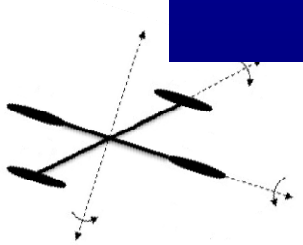


```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

i = 5, j = 0
list[i]를 result[j]에 저장





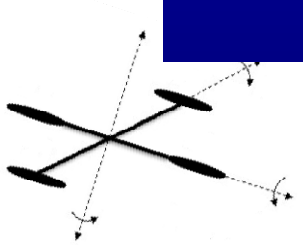
```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

수행 한 다음, i는 6,
j는 -1이 됨

list	1	2	3	4	5	6

newList	6	5	4	3	2	1



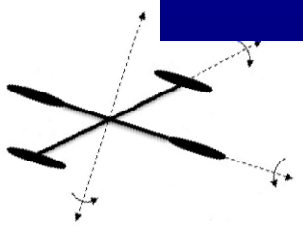
```
int list1[] = {1, 2, 3, 4, 5, 6};
reverse(list1, list2);
```

```
void reverse(const int list[], int newList[], int size)
{
  for (int i = 0, j = size - 1; i < size; i++, j--)
  {
    newList[j] = list[i];
  }
}
```

i (=6) < 6은 false. 루프를 마침

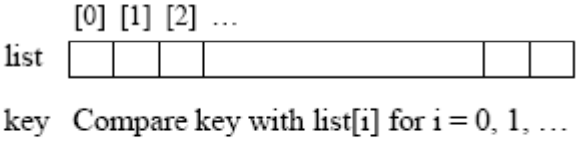
list	1	2	3	4	5	6

newList	6	5	4	3	2	1



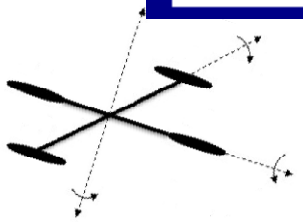
- 선형탐색

```
int linearSearch(int list[], int key, int arraySize)
{
  for (int i = 0; i < arraySize; i++)
  {
    if (key == list[i])
      return i;
  }

  return -1;
}
```

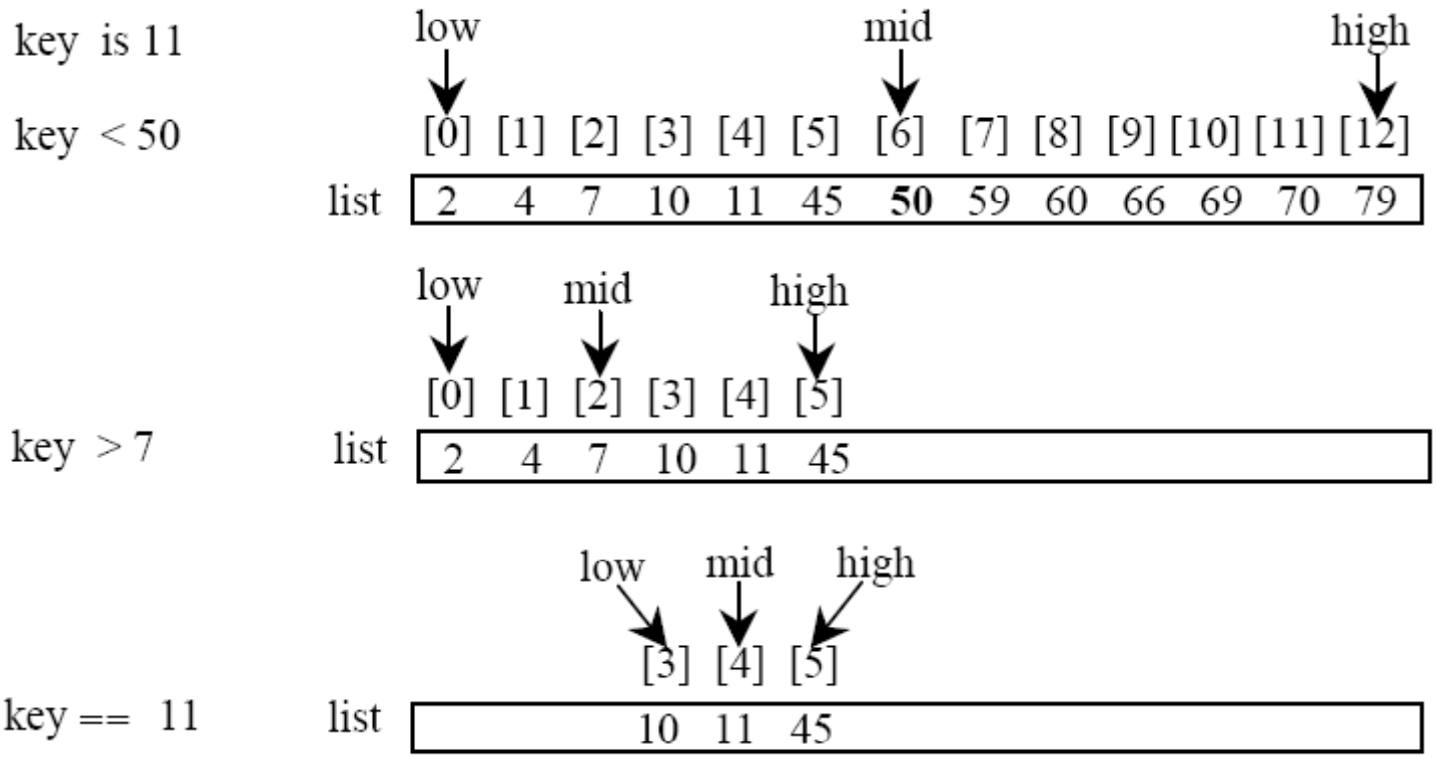


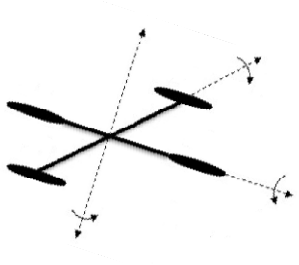
함수 확인

```
int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
int i = linearSearch(list, 4);  // returns 1
int j = linearSearch(list, -4); // returns -1
int k = linearSearch(list, -3); // returns 5
```



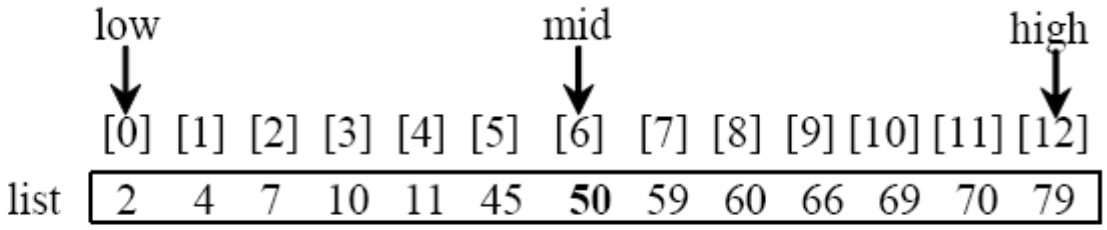
- 이진탐색


key is 11
low
mid
high
key < 50
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12]
list
2 4 7 10 11 45 50 59 60 66 69 70 79
low
mid
high
[0] [1] [2] [3] [4] [5]
key > 7
list
2 4 7 10 11 45
low
mid
high
[3] [4] [5]
key == 11
list
10 11 45

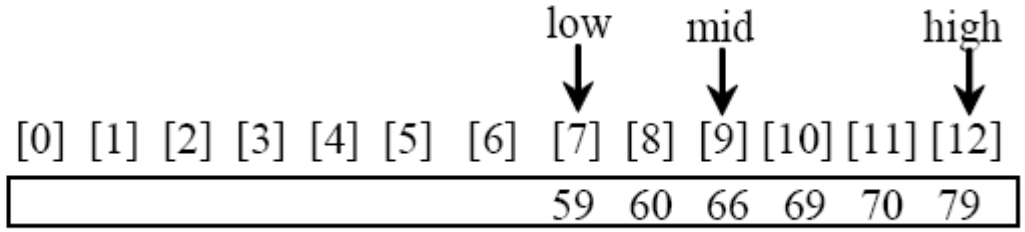


key is 54

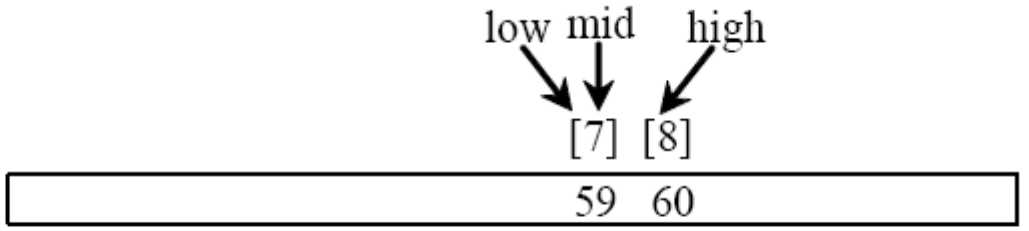
key > 50

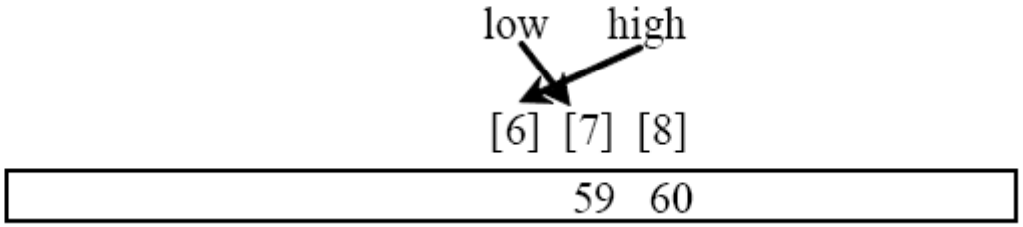


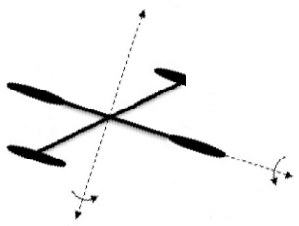
key < 66



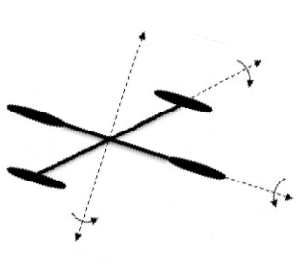
key < 59



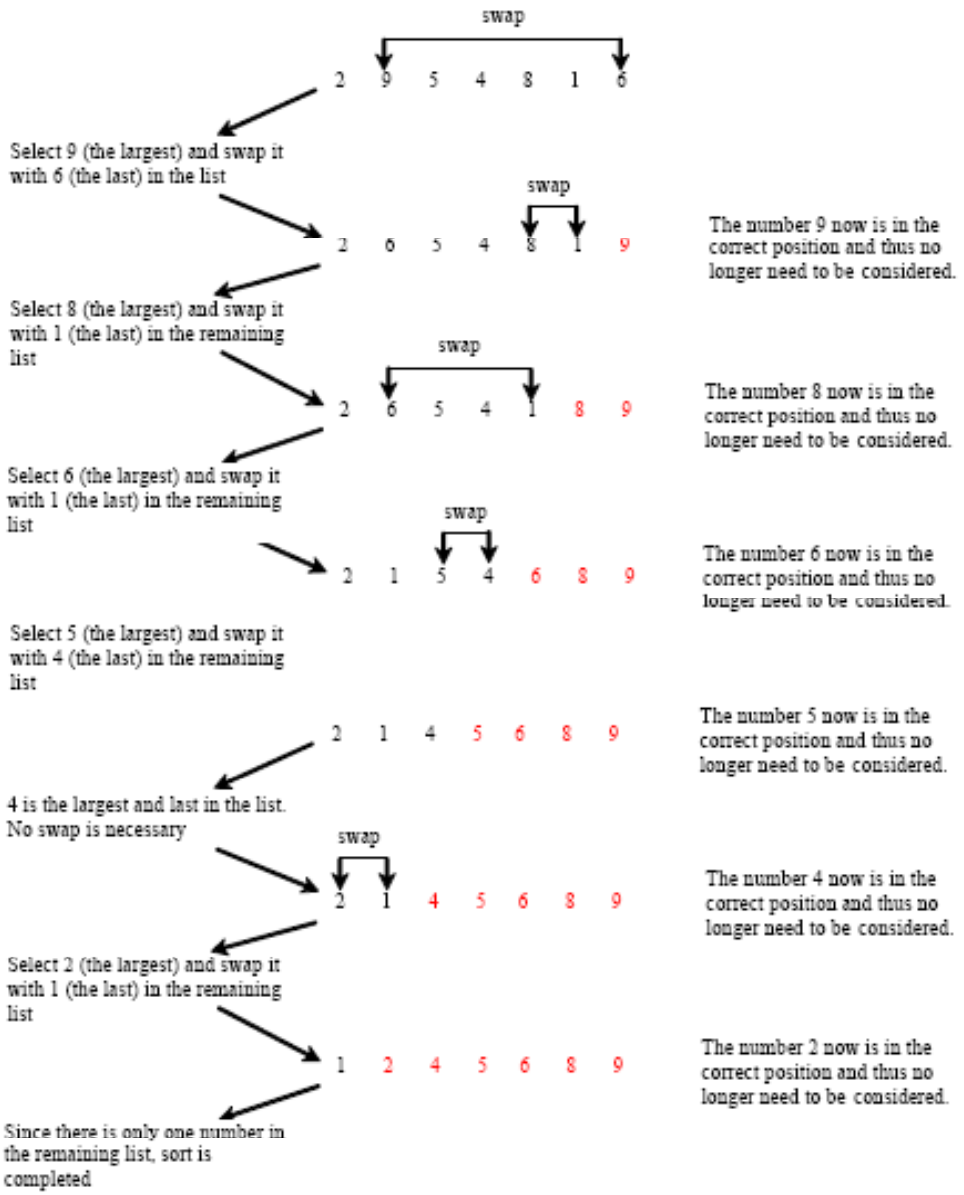


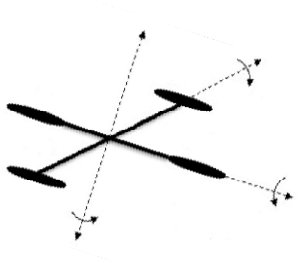


```
int binarySearch(int list[], int key, int arraySize)
{
 int low = 0;
 int high = arraySize - 1;

 while (high >= low)
 {
  int mid = (low + high) / 2;
  if (key < list[mid])
   high = mid - 1;
  else if (key == list[mid])
   return mid;
  else
   low = mid + 1;
 }

 return –low - 1;
}
```

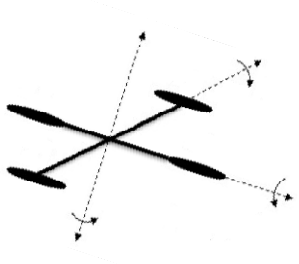


- 배열의 정렬 (선택정렬)

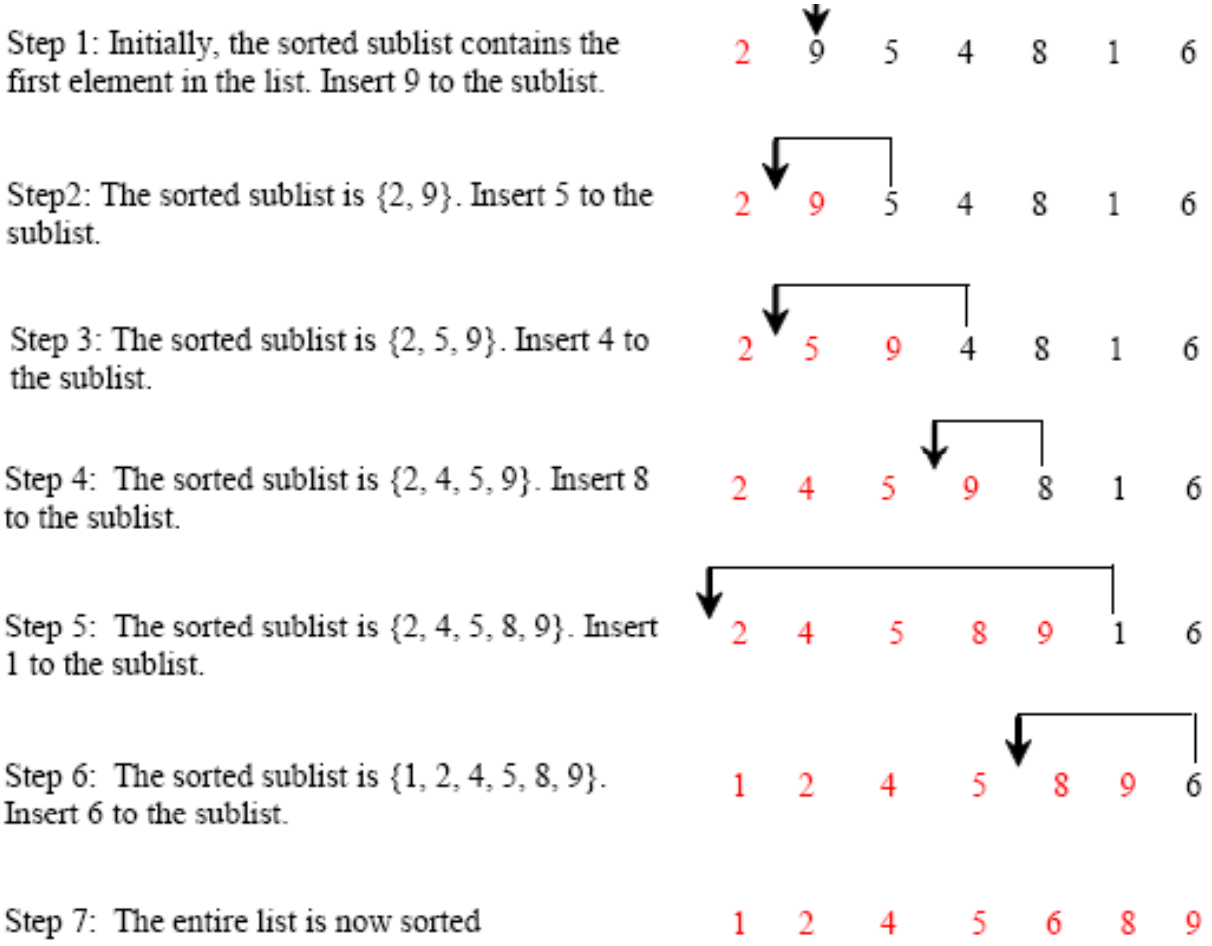

swap
2 9 5 4 8 1 6
Select 9 (the largest) and swap it with 6 (the last) in the list
swap
2 6 5 4 8 1 9
The number 9 now is in the correct position and thus no longer need to be considered.
Select 8 (the largest) and swap it with 1 (the last) in the remaining list
swap
2 6 5 4 1 8 9
The number 8 now is in the correct position and thus no longer need to be considered.
Select 6 (the largest) and swap it with 1 (the last) in the remaining list
swap
2 1 5 4 6 8 9
The number 6 now is in the correct position and thus no longer need to be considered.
Select 5 (the largest) and swap it with 4 (the last) in the remaining list
2 1 4 5 6 8 9
The number 5 now is in the correct position and thus no longer need to be considered.
4 is the largest and last in the list. No swap is necessary
swap
2 1 4 5 6 8 9
The number 4 now is in the correct position and thus no longer need to be considered.
Select 2 (the largest) and swap it with 1 (the last) in the remaining list
1 2 4 5 6 8 9
The number 2 now is in the correct position and thus no longer need to be considered.
Since there is only one number in the remaining list, sort is completed

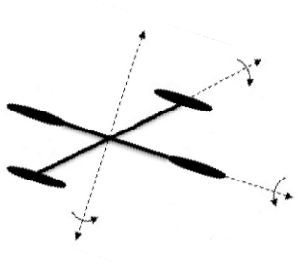


```
void selectionSort(double list[], int arraySize)
{
  for (int i = arraySize - 1; i >= 1; i--)
  {
    // Find the maximum in the list[0..i]
    double currentMax = list[0];
    int currentMaxIndex = 0;

    for (int j = 1; j <= i; j++)
    {
      if (currentMax < list[j])
      {
        currentMax = list[j];
        currentMaxIndex = j;
      }
    }

    // Swap list[i] with list[currentMaxIndex] if necessary;
    if (currentMaxIndex != i)
    {
      list[currentMaxIndex] = list[i];
      list[i] = currentMax;
    }
  }
}
```

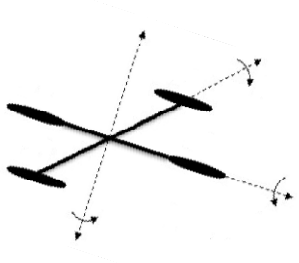


- 배열정렬 (삽입 정렬)


Step 1: Initially, the sorted sublist contains the first element in the list. Insert 9 to the sublist.
2 9 5 4 8 1 6
Step2: The sorted sublist is {2, 9}. Insert 5 to the sublist.
2 9 5 4 8 1 6
Step 3: The sorted sublist is {2, 5, 9}. Insert 4 to the sublist.
2 5 9 4 8 1 6
Step 4: The sorted sublist is {2, 4, 5, 9}. Insert 8 to the sublist.
2 4 5 9 8 1 6
Step 5: The sorted sublist is {2, 4, 5, 8, 9}. Insert 1 to the sublist.
2 4 5 8 9 1 6
Step 6: The sorted sublist is {1, 2, 4, 5, 8, 9}. Insert 6 to the sublist.
1 2 4 5 8 9 6
Step 7: The entire list is now sorted
1 2 4 5 6 8 9



```
void insertionSort(double list[], int arraySize)
  {
  for (int i = 1; i < arraySize; i++)
  {
    /* insert list[i] into a sorted sublist list[0..i-1] so that
       list[0..i] is sorted. */
    double currentElement = list[i];
    int k;
    for (k = i - 1; k >= 0 && list[k] > currentElement; k--)
    {
      list[k + 1] = list[k];
    }

    // Insert the current element into list[k+1]
    list[k + 1] = currentElement;
  }
}
```



- 2차원 배열

	[0]	[1]	[2]	[3]	[4]
[0]					
[1]					
[2]					
[3]					
[4]					

```
int matrix[5][5];
```

	[0]	[1]	[2]	[3]	[4]
[0]					
[1]					
[2]		7			
[3]					
[4]					

```
matrix[2][1] = 7;
```

	[0]	[1]	[2]	[3]
[0]	1	2	3	
[1]	4	5	6	
[2]	7	8	9	
[3]	10	11	12	

```
int array[][] = {
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9},
   {10, 11, 12}
};
```

```
int array[4][3] =
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9},
   {10, 11, 12}
};
```

(a)

Equivalent

```
int array[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

(b)