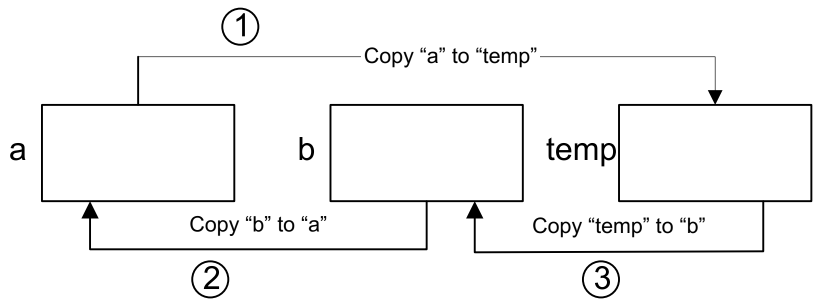
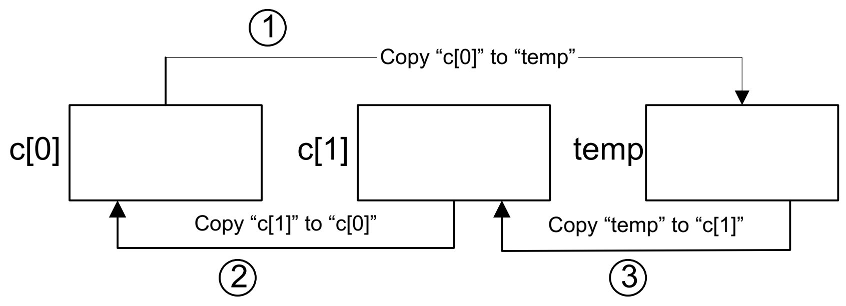
The act of swapping two variables refers to mutually exchanging the values of the variables. This can be done with or without temporary variables. The following shows swapping being done using a temp variable.

```
int main( )
{
      int  a=10,
           b=20,
           temp;

      int c[2]={10,20};

      //Swap the contents of "a" and "b"
      temp=a;
      a=b;
      b=temp;

      //Swap the contents of array
      temp=c[0];
      c[0]=c[1];
      c[1]=temp;
}
```

Swap





Sorting

- Sorting algorithms are used to arrange random data into some order.

    - Ascending order
      Values in the array are stored from lowest to highest.

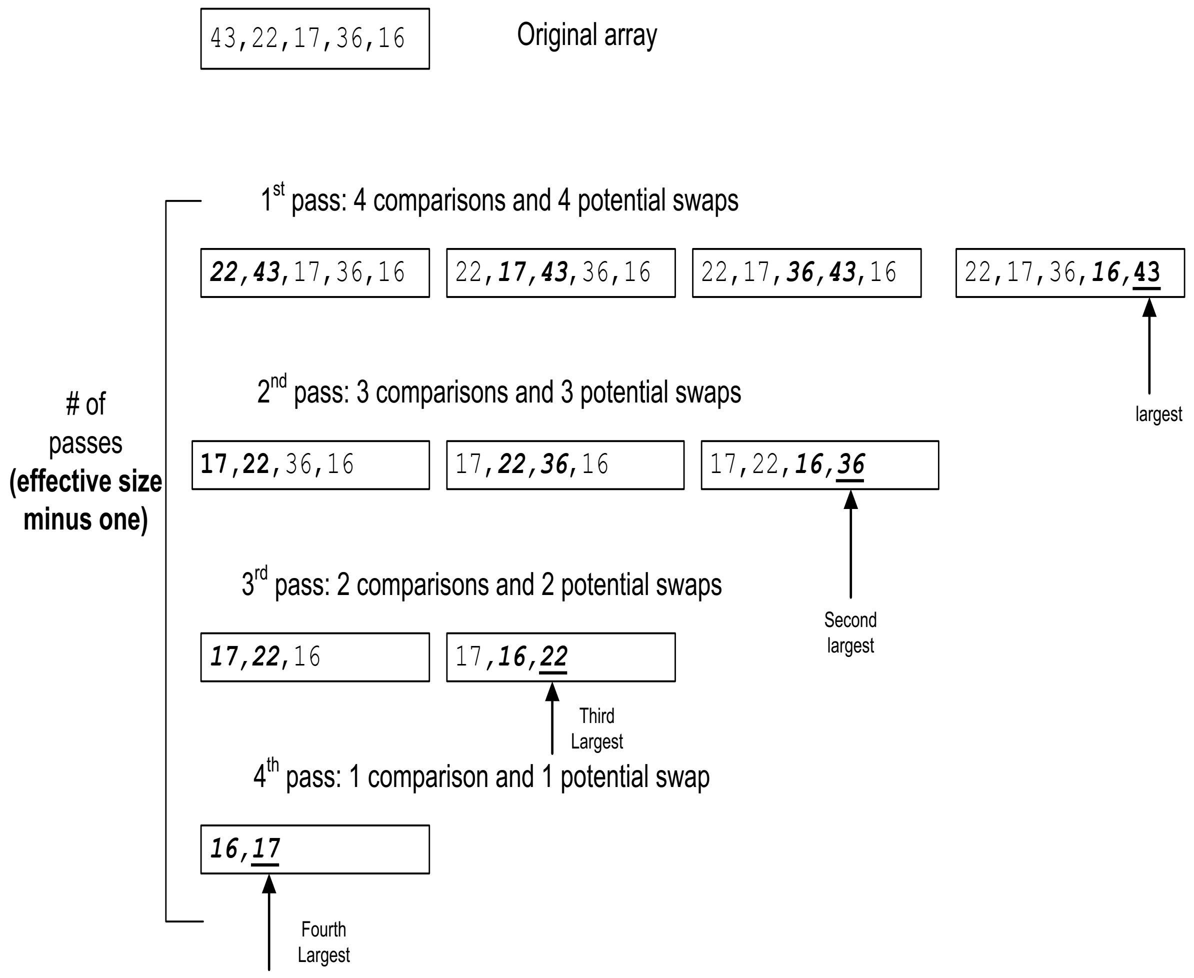
    - Descending order
      Values in the array are stored from highest to lowest.

- **Selection sort** repeatedly finds the next largest (or smallest) element in the array and moves it to its final position.

- **Bubble sort** is the simplest and also the slowest. The basis of this algorithm is to compare each element in the list with the element next to it, and swap them if required. If sorting in ascending order, the larger values "bubble" to the end of the list while smaller values "sink" towards the beginning of the list. If sorting in descending order, the smaller values "bubble" to the end of the list while larger values "sink" towards the beginning of the list.

- **Insertion sort** is done by iterating up the array, growing the sorted list behind it. At each array-position, it checks the value there against the largest value in the sorted list (which happens to be next to it, in the previous array-position checked). If larger, it leaves the element in place and moves to the next. If smaller, it finds the correct position within the sorted list, shifts all the larger values up to make a space, and inserts into that correct position.

Bubble Sort


43,22,17,36,16
Original array
1st pass: 4 comparisons and 4 potential swaps
22,43,17,36,16
22,17,43,36,16
22,17,36,43,16
22,17,36,16,43
largest
# of passes (effective size minus one)
2nd pass: 3 comparisons and 3 potential swaps
17,22,36,16
17,22,36,16
17,22,16,36
Second largest
3rd pass: 2 comparisons and 2 potential swaps
17,22,16
17,16,22
Third Largest
4th pass: 1 comparison and 1 potential swap
16,17
Fourth Largest

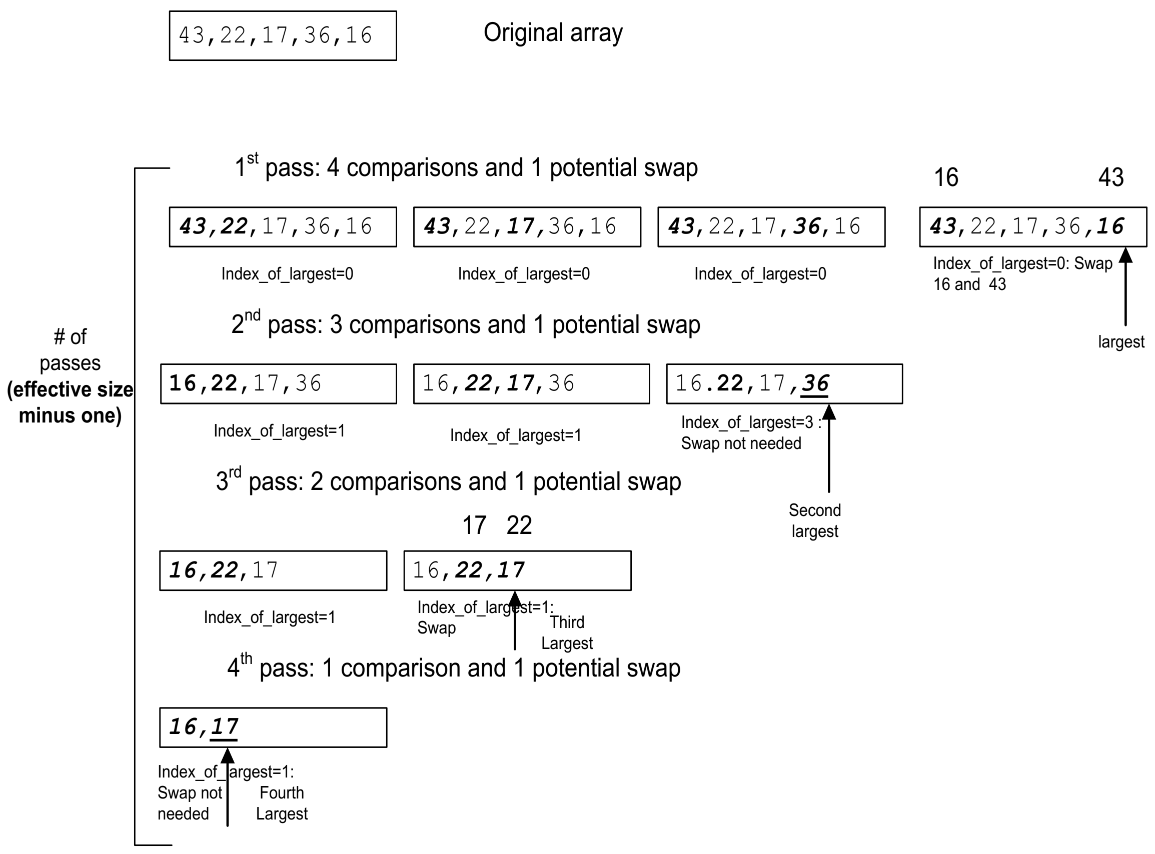
<u># of Passes</u>: Equivalent to the effective size minus one (4)

- **<u>Pass 1</u>**: 4 comparisons and 4 potential swaps puts largest number in position
- **<u>Pass 2:</u>** 3 comparisons and 3 potential swaps puts the second largest number in position
- **<u>Pass 3:</u>** 2 comparisons and 2 potential swaps puts the third largest number in position
- **<u>Pass 4:</u>** 1 comparison and 1 potential swap puts the fourth largest number in position

BIG O: O(N^2)

```
#include <iostream>
using namespace std;

#define SIZE 10

void bubble(int arr[SIZE], int limit)
{
    int temp, index;

    //This loop is used to determine the number of passes
    for (;limit > 0;limit--){
      for (index=0; index<limit; index++)
      {
            //To change to descending order just change the
            //relational operator to <
            if (arr[index] > arr[index+1])
            {
                  //Swap array element
                  temp=arr[index];
                  arr[index]=arr[index+1];
                  arr[index+1]=temp;
            }
      }
   }
}

int main()
{
      int arr[SIZE] ={43,22,17,36,16} ;
      int effective_size=5;

      bubble(arr,effective_size-1); //Pass entire array
      for (int i=0; i<effective_size;i++)
            cout<<arr[i]<<endl;
}
```

Selection Sort



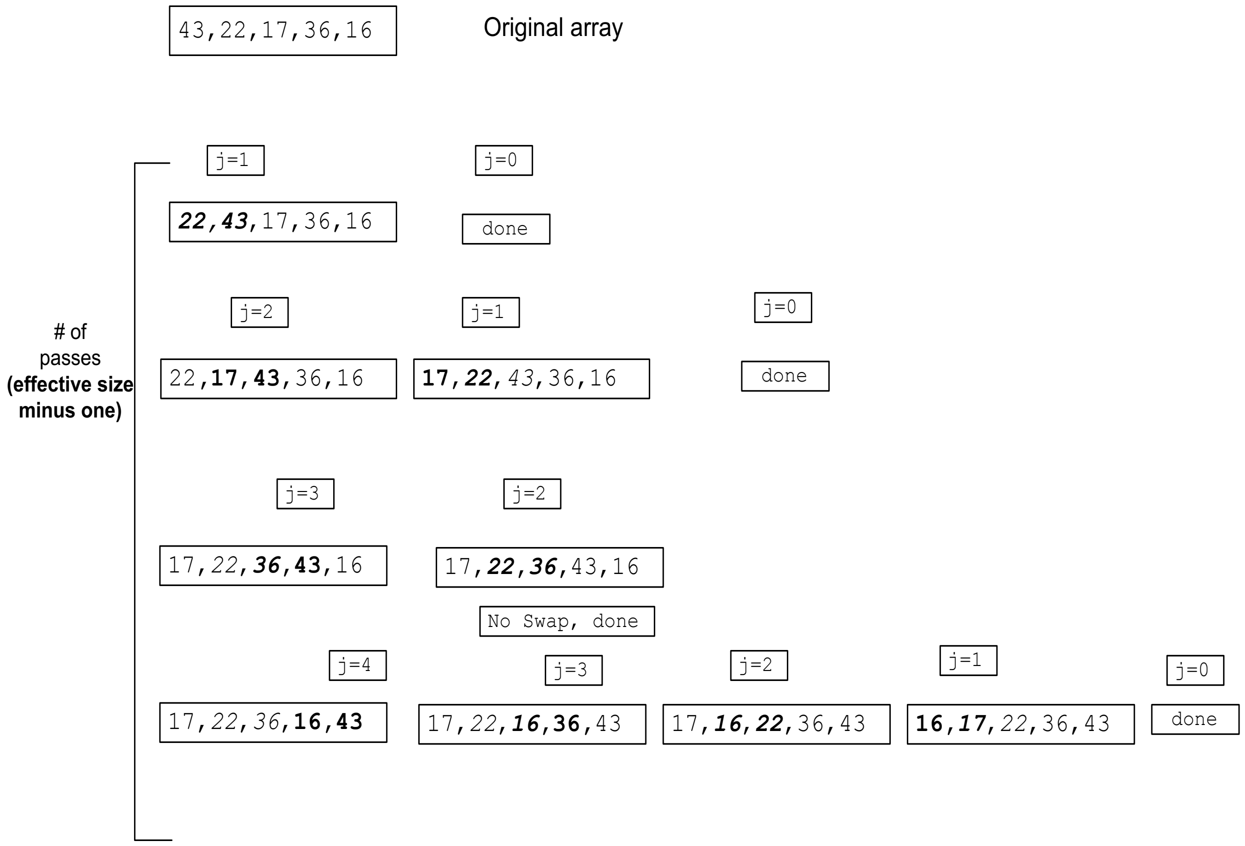
# of Passes: Equivalent to the effective size minus one (4)

- **Pass 1**: 4 comparisons and 1 potential swap puts largest number in position
- **Pass 2:** 3 comparisons and 1 potential swap puts the second largest number in position
- **Pass 3:** 2 comparisons and 1 potential swap puts the third largest number in position
- **Pass 4:** 1 comparison and 1 potential swap puts the fourth largest number in position

BIG O: O(N^2)

```
#include <iostream>
using namespace std;

#define SIZE 10

void selection(int arr[], int limit)
{
      int temp, index_of_largest,index;

      //This loop is used to determine the number of passes
      for(;limit > 0;limit--){
            index_of_largest=0 ;

            //This loop is used to determine the number of
            //comparisons for each pass
            for (index=1; index<=limit; index++) {

                  //To change to descending order just change the
                  //relational operator to <.
                  if (arr[index] > arr[index_of_largest])
                        index_of_largest=index; //Store the
                                                 //index of
                                                 //array element
            }

            //Swap element at the end of pass if needed
            if (limit !=index_of_largest){
                  temp=arr[limit];
                  arr[limit]=arr[index_of_ largest];
                  arr[index_of_ largest]=temp;
            }
      }
}

int main()
{
      int arr[SIZE] ={43,22,17,36,16} ;
      int effective_size=5;

      selection(arr,effective_size-1);    //Pass entire array
      for (int i=0; i<effective_size; i++){
            cout << arr[i]<<endl;
      }
}
```

Insertion Sort



<u># of Passes</u>: Equivalent to the effective size minus one (4)
Variable number of comparisons on each pass

BIG O: O(N^2)

```
#include <iostream>
using namespace std;
#define SIZE 6

void print_array(int arr[])
{
      cout<< "insertion sort steps: ";
      for (int i=0; i<SIZE; i++)
            cout<<arr[i]<<" ";
      cout<<endl;
}

void insertion_sort(int arr[])
{
      int i, j ,tmp;

      for (i = 1; i < SIZE; i++)  {
            for (j=i; j>0 && arr[j] < arr[j-1];j--){
                  tmp = arr[j];
                  arr[j] = arr[j - 1];
                  arr[j - 1] = tmp;
            }
      }
}

int main(){
      int a[]={43,22,17,36,16} ;

      insertion_sort(a);
      print_array(a);
}
```

Shell sort

Shellsort is a generalization of insertion sort that allows the exchange of items that are far apart.

```
Loop (different gaps) n/2  as long as gap >=1   (gap)
   Loop  (scan the array from the gap to the end of the array)  I as long as less than array size (i)
          Loop(Compare the elements within gap)  as long as j>=gap and elements in wrong place
                    Swap
```

8,7,6,5,4,3,2,1 Original array

Gap=4

Subtract by gap

j=4 **4**,7,6,5,**8**,3,2,1	j=0 DONE **j>=gap** and a[j-gap]> a[j]

j=5 4,**3**,6,5,8,**7**,2,1	j=1 DONE **j>=gap** and a[j-gap]> a[j]

j=6 4,3,**2**,5,8,7,**6**,1	j=2 DONE **j>=gap** and a[j-gap]> a[j]

j=7 4,3,2,**1**,8,7,6,**5**	j=3 DONE **j>=gap** and a[j-gap]> a[j]

Gap=2

j=2 **2**,3,**4**,1,8,7,6,5	j=0 DONE **j>=gap** and a[j-gap]> a[j]

j=3 2,**1**,4,**3**,8,7,6,5	j=1 DONE **j>=gap** and a[j-gap]> a[j]

j=4 2,1,**4**,3,**8**,7,6,5 DONE j>=gap and **a[j-gap]> a[j]**

j=5 2,1,4,**3**,8,**7**,6,5 DONE j>=gap and **a[j-gap]> a[j]**

j=6 2,1,4,3,6,7,8,5	j=4 2,1,**4**,3,**6**,7,8,5 Done j>=gap and **a[j-gap]> a[j]**

j=7 2,1,4,3,6,**5**,8,**7**	j=5 2,1,4,**3**,6,**5**,8,7 Done j>=gap and **a[j-gap]> a[j]**

Gap=1

j=1 **1,2**,4,3,6,5,8,7	j=0 **Done j>=gap** and a[j-gap]> a[j]

j=2 1,**2**,**4**,3,6,5,8,7 Done j>=gap and **a[j-gap]> a[j]**

j=3 1,2,**3**,**4**,6,5,8,7	j=2 1,**2**,**3**,4,6,5,8,7 Done j>=gap and **a[j-gap]> a[j]**

j=4 1,2,3,**4**,**6**,5,8,7 Done j>=gap and **a[j-gap]> a[j]**

j=5 1,2,3,4,**5**,**6**,8,7	j=4 1,2,3,**4**,**5**,6,8,7 Done j>=gap and **a[j-gap]> a[j]**

j=6 1,2,3,4,5,**6**,8,7 Done j>=gap and **a[j-gap]> a[j]**

j=7 1,2,3,4,5,6,**7**,**8**	**j=6** 1,2,3,4,5,**6**,**7**,8 Done j>=gap and **a[j-gap]> a[j]**

```
#include <iostream>
using namespace std;
void display(int a[], int n){
      for (int i=0; i<n; i++)
            cout<<a[i]<<" ";
}

void shell(int a[], int n){
      int temp;
      for (int gap=n/2; gap>=1; gap=gap/2)
            for(int i=gap; i<n; i++)
                  for (int j=i; j>=gap && a[j-gap]>a[j];j=j-gap){
                        temp=a[j-gap];
                        a[j-gap]=a[j];
                        a[j]=temp;
                  }
      display(a,n);
}

int main(){
      int a[]={8,7,6,5,4,3,2,1};
      shell(a,8);
}
```

Inversion Count

```
//Inversion Count for an array indicates – how far (or close)
//the array is from being sorted. If array is already sorted
//then inversion count is 0. If array is sorted in reverse order
//that inversion count is the maximum.
//The sequence 2, 4, 1, 3, 5 has three inversions
//(2, 1), (4, 1), (4, 3).
// 2 is only greater than 1, 4 is greater than 1 and 3,
//1 and 3 are good, 5 there is nothing after it
//Can be used to see if your array is sorted if inversion count=0
//Also lower numbers is good for ascending order whereas higher
//numbers are good for descending order sorts in terms of
//efficieny
/* you have two people who have come up with a set of rankings
and we want to compare it with the expert ranking. They both get
it wrong but we want to find out who is closer */
```

Lowest infant mortality rates

Expert		Player1		Player 2	
Finland japan Portugal Sweden Czech	1 2 3 4 5	Portugal Finland Japan Czech Sweden	3 1 2 5 4	Czech Finland japan Portugal Sweden	5 1 2 3 4
inversion	0		3		4

BIG O: O(n log n)

```
#include <cstdio>
#include <iostream>
using namespace std;

int getInvCount(int arr[], int n)
{
  int inv_count = 0;
  int i, j;

  //Start from the left and compare each one against
  //everything on its right
  //Does not need to check the last item because there is
  //nothing after it.
  for(i = 0; i < n-1; i++)
    for(j = i+1; j < n; j++)  //start with one after the item that is checked.
      if(arr[i] > arr[j])
        inv_count++;

  return inv_count;
}

int main()
{
  int arr[] = {1, 20, 6, 4, 5};
  printf(" Number of inversions are %d \n", getInvCount(arr, 5));
  int a;
  cin>>a;
}
```

Searching

- Sequential search is the simplest algorithm. It tests a key against each element in the list successively. The list does not need to be sorted. However, in the worst case all elements might have to be tested.

Sequential Search
BIG O: O(N)

```
#include <iostream>
using namespace std;

#define SIZE 5

bool search(int searchKey, int a[]);

int main( ){
      int a[SIZE]={5,9,11,12,14};
      int searchKey;

      cout << "what number do you want to search for";
      cin>>searchKey;
      if (search(searchKey, a))
            cout << "Found";
      else
            cout <<"Not found";
}

bool search(int searchKey, int a[]){

      bool found=false;

      for (int i=0; i<SIZE; i++)     //Scan every element in list
            if (a[i]==searchKey){    //If found then break out of
                  found=true;        //loop
                  i=SIZE;
      }
      return(found);
}
```

Binary Search

- Binary search works on a sorted list. It tests whether the array element (halfway between positions low and high) is equal to the key. If the element is equal to key, then the search is successful. If the array element at the position tested is too large, then change the value of high to one less than the tested position. If the array element at the position tested is too small, then change the value of low to one more than the tested position. Continue this procedure as long as key is not found and low is less than or equal to high.

The line to calculate the mean of two integers:

```
      mid = (low + high) / 2
```

could produce the wrong result in some programming languages when used with a bounded integer type, if the addition causes an overflow. (This can occur if the array size is greater than half the maximum integer value.) If signed integers are used, and low + high overflows, it becomes a negative number, and dividing by 2 will still result in a negative number. Indexing an array with a negative number could produce an out-of-bounds exception, or other undefined behavior. If unsigned integers are used, an overflow will result in losing the largest bit, which will produce the wrong result.

One way to fix it is to manually add half the range to the low number:

```
      mid = low + (high - low) / 2
```

BIG O: O(Log N)

```
#include <iostream>
using namespace std;

#define SIZE 7

int binary_search(int A[], int key)
{
  int high = SIZE-1, low = 0, mid;
  bool found=false;

  while (high >= low && !found){
      // calculate the midpoint for roughly equal partition
      mid = (high + low ) / 2;

      if (key>A[mid] )
          low = mid + 1;
      else if (key<A[mid])
           high = mid - 1;
      else
           found=true;
  }
  return found;
}

int main(){

      int a[]={5,9,11,12,13,15,18,20}, searchKey;

      cout << "\nwhat number do you want to search for? ";
      cin>>searchKey;
      if (binary_search(a,searchKey))
            cout << "Found";
      else
            cout <<"Not found";
}
```

KEY=19 `mid = (high + low ) / 2;`

SUBSCRIPT	0	1	2	3	4	5	6	7
Array	5	9	11	12	13	15	18	20

1	L			M				H
2					L	M		H
3							L M	H
4								LHM
5							H	L
6								

KEY=11 `mid = (high + low ) / 2;`

SUBSCRIPT	0	1	2	3	4	5	6	7
Array	5	9	11	12	13	15	18	20

1	L			M				H
2	L	M	H					
3			LHM					
4								
5								
6								

KEY=18 `mid = low + (high - low) / 2`

SUBSCRIPT	0	1	2	3	4	5	6
Array	5	9	11	12	13	15	18

1	L			M			H
2					L	M	H
3							LHM
4							
5							
6							

KEY=11 `mid = low + (high - low) / 2`

SUBSCRIPT	0	1	2	3	4	5	6
Array	5	9	11	12	13	15	18

1	L			M			H
2	L	M	H				
3			LHM				
4							
5							
6							