Binary Search
and
Selection Sort

In this section we shall take up two representative problems in computer science, work out the algorithms based on the best strategy to solve the problems. The two problems, one related to searching and the other related to data sorting are very common in many real world situations.

Binary Search Method:

Suppose we are given a number of integers stored in an array \( A \), and we want to locate a specific target integer \( K \) in this array. If we do not have any information on how the integers are organized in the array, we have to sequentially examine each element of the array. This is known as linear search.

The basic idea is to examine each element of the array starting with the element at the first index value, and going to the last value of the index. If the element matches the target \( K \), then search can terminate successfully, by returning the index of the element in the array. If all elements are exhausted and the target is not found in the array, then the function returns a special value such as \(-1\).

```c
int LinearSearch (int A[ ], int n, int TARGET) {
    int i;
    for (i = 0; i < n; i++)
    {
        if ( A[i] == TARGET)
            return i;
    }
    return -1 ;
}
```
However, if the elements of the array are ordered, let us say in ascending order, and we wish to find out the position of an integer target \( \text{TARGET} \) in the array, we need not make a sequential search over the complete array. We can make a faster search using the **Binary search method**.

The basic idea is to start with an examination of the middle element of the array. If this matches the target \( \text{TARGET} \), then search can terminate successfully, by returning the index of the element in the array. On the other hand, if \( \text{TARGET} < \text{A[middle]} \), then search can be limited to the elements to the left of \( \text{A[middle]} \). All elements to the right of middle index can be ignored.

If it turns out that \( \text{TARGET} > \text{A[middle]} \), then further search is limited to elements to the right of \( \text{A[middle]} \). If all elements are exhausted and the target is not found in the array, then the function returns a special value such as \(-1\).

Here is one version of the Binary Search function:

```c
int BinarySearch (int A[ ], int n, int TARGET)
{
    int L=0, Mid, R= n-1;
    while (L<=R)
    {
        Mid = (L +R)/2;
        if ( TARGET == A[Mid] )
            return Mid;
        else if ( TARGET > A[Mid] )
            L = Mid + 1;
        else
            R = Mid - 1 ;
    }
    return -1 ;
}
```

We shall formally compare the relative time taken by the two search techniques when we study time complexity.
Selection Sort

Selection sort is an attempt to localize the exchanges of array elements by finding a misplaced element first and putting it in its final place.

⇒ The list is divided into two sublists, *sorted* and *unsorted*, which are divided by an imaginary wall.

⇒ We select the smallest element from the unsorted sublist and swap it with the element at the beginning of the unsorted data.

⇒ Then the smallest value among the remaining elements is selected and put in the second position and so on.

⇒ After each selection and swapping, the imaginary wall between the two sublists move one element ahead, increasing the number of sorted elements and decreasing the number of unsorted ones.

⇒ Each time we move one element from the unsorted sublist to the sorted sublist, we say that we have completed a sort pass.

⇒ A list of \( n \) elements requires \( n-1 \) passes to completely rearrange the data.
Selection Sort Example

Original List

Sorted

23  78  45  8  32  56

Unsorted

After pass 1

8  78  45  23  32  56

After pass 2

8  23  45  78  32  56

After pass 3

8  23  32  78  45  56

After pass 4

8  23  32  45  78  56

After pass 5

8  23  32  45  56  78
Selection Sort Algorithm

for cur = 0 to cur = n
    Start with cur being named the index to smallest element.
    go through the remaining array
    find smallest the index to smallest element.
    swap value at smallest with the value at cur.

Selection Sort Function

/* Sorts by selecting smallest element in unsorted
portion of array and exchanging it with element
at the beginning of the unsorted list. */

void selectionSort(int list[], int n)
{
    int cur, j, smallest, temp;

    for (cur = 0; cur <= n; cur ++){
        smallest = cur;

        for ( j = cur +1; j <= n ; j++)
            if(list[ j ] < list[smallest])
                smallest = j ;

        // Smallest selected; swap with current element
        temp  = list[cur];
        list[cur] = list[smallest];
        list[smallest] = temp;
    }
}