Data Structures and Algorithms — Lab I and Lab 2

Objective

- Revision Basic concepts of array
- Introduction to *Attributes* and *Methods*
- Simple sorting algorithms — Selection sort
- Simple searching algorithms — Linear search, binary search
- Working with classes

Array

Pracrtice Task (Revision)

Develop a C++ function that accepts int array & char array and the array's size as arguments. An array, which contains the mix of name and number. Make sure to regrow your array. **Submission without implementation of REGROW FUNCTIONS will be not acceptable.**

- Given an array ;

  Array[]={"A0h12m34a56dK78han"}

- two functions, which can copy each content in separate arrays (integer into int array and characters into char array with respect to it's sizes dynamically memory allocation).

- These functions should return pointer to the new array. (dynamic array)

Expected Output:

Name : AhmadKhan

Number : 012345678

Sorting

A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of element in the respective data structure.

Sorting Algorithms : Following are some Sorting algorithms:

- Selection Sort
- Bubble Sort
- Insertion Sort
- Merge Sort etc

Selection Sort

Selection sort is a sorting algorithm, in which we repeatedly find the next largest (or smallest) element in the array and move it to its final position in the sorted array. Assume that we wish to sort the array in increasing (ascending) order, i.e. the smallest element at the beginning of the array and the largest element at the end. We begin by selecting the smallest element and moving it to the lowest index position. We can do this by swapping the element at the lowest index and the smallest element. We then reduce the effective size of the array by one element and repeat the process on the greater (sub)array. The process stops when the effective size of the array becomes 1 (an array of 1 element is already sorted).

Pseudo code

```
Input: An unsorted array, A of N elements
Output: Sorted array, A
For I = 0:N-1
    SmallSub = I
    For J = I+1:N-1
       If A[J] < A[SmallSub]
          SmallSub = J
       End-If
    End-For
    Swap ( A[I], A[SmallSub] )
End-For
```

Searching Algorithms

Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored. Based on the type of search operation, these algorithms are generally classified into two categories:

1. **Sequential Search**: In this, the list or array is traversed sequentially and every element is checked. For example: Linear Search.
2. **Interval Search**: These algorithms are specifically designed for searching in sorted data-structures. These type of searching algorithms are much more efficient than Linear Search as they repeatedly target the center of the search structure and divide the search space in half. For Example: Binary Search.

Linear Search

In computer science, linear search or sequential search is a method for finding a particular value in a list that consists of checking every one of its elements, one at a time and in sequence, until the desired one is found.

Pseudo code

```
Input: An unsorted array, A of N elements and value to be searched
Output: Index of searched element or -1 if not found

For I = 0:N-1
   If ( A[I] == Value )
     Return [I];
   End-if
End-For
return -1;
```

Binary Search

Binary search is another searching algorithm, used to search a specific value (or index of value) from the *sorted array*.

In binary search, we first compare the *value to be searched* with the item in the middle position of the array. If there's a match, we can return immediately. If the key is less than the middle key, then the item sought must lie in the lower half of the array; if it's greater than the item sought must lie in the upper half of the array. So we repeat the procedure on the lower (or upper) half of the array.

Pseudo Code

```
Input: An Sorted array, A of N elements and value to be searched
Output: Index of searched element or -1 if not found

low = 0, high = N-1;
while (low <= high)
   mid = (low + high)/2;
   If ( A[mid] == Value )
     Return mid;
   Else-if ( A[mid] < Value )
     low = mid + 1;
   Else
     high = mid - 1;
   End-if
End-For
return -1;
```

Tasks

Students are required to complete the following tasks in lab timings.

Task 1

Implement the Selection Sort Algorithm. And sort the following array.(Selection Sort)

```
Input: 64 25 12 22 11


Output: 11 12 22 25 64
```

Task 2

Write a function to find an element “a” in given array that is array[] of n elements.(Linear Search)

```
Input : array[] = {100, 200, 800, 300, 600,
                   450, 120, 10, 30, 70}
          a = 10;
Output : 7
Element a is present at index 7
```

Task 3

Write a function to find an element “a” in given array that is array[] of n elements.(Binary Search)

Note: Array should be sorted before implementing binary search algorithm

```
Input : array[] = {100, 200, 800, 300, 600,
                   450, 120, 10, 30, 70}
          a = 100;
Output :
Sorted array :
array[]={10,30,70,100,120,200,300,450,600,800} 3
Element a is present at index 3
```

Task 4

Compute the complexity of the following codes, according to their line by line execution. Finally compute their big(O) complexity.

a.

```
void phytagorean(int value)
{
for(int i = 1; i <= value; i++)
{
        for(int j = 1; j <= value; j++)
        {
                for(int k = 1; k <= value; k++)
                {
                int num1 = (i*i) + (j*j);
                int num2 = (k*k);
                if (num1 == num2)
                        cout<<"Pair is: (" << i << ", " << j << ", " << k << ")" << endl;
                }
         }
}
}
```

b.

```
void RangeCheck(int arr[],int sixe, int num1, int num2){
   int counter = 0;
```

```
    for (int i=0; i<sixe; i++)
    {
       if (arr[i] >= num1 && arr[i] <= num2)
       {
          counter++;
          cout << arr[i] << "is in the range " << endl;
       }
    }
    cout << "Total Numbers in range are: " << counter << endl;
}
```

Task 5

Implement an abstract class template "Array" (Header file i.e Array.h) that contains three data members: An array of **Template type,** an integer variable of **current size** and an integer variable of **maximum size** of the array.

Create Interface (Array.h) of the Template class "Array" header which has only two pure virtual functions for adding and removing item from top of the array and display data members value of the class in implementation (where Array.h will be inherited for implementing the functionalities i.e myArray.h), use arguments for template type in implementation and interface (passing data between interface i.e Array.h and impelementation i.e myarray.h).

Add two additional functionalities of removing from specific index, adding at specific index to the class "Array".

Hint: You may pass arguments from the driver code (where your code for main function exists i.e main.cpp) to interface for implementation while creating an object.

For your help below is an image attached for writing driver code for this very task:

```
#include "myarray.h"

int main()
{
    myArray<char> obj(6);
    obj.addItem('A');
    obj.addItem(30);
    obj.addItem(70);

    obj.display();

    obj.remItem();
    obj.remItem();

    obj.display();
    system("pause");

    return 0;
}
```

Bonus Task

Add a Menu driven code for selecting between implemented functionalities of:

- Adding at top.
- Removing from top.
- Adding at specific index within maximum limit.
- Removing at specific index within maximum limit.