CSE1033 – Data Structure

Assignment I

Submission To:
Md. Siyam Sajeeb Khan (MSSK)
Lecturer & Coordinator, Dept of CSE
Southeast University

Submitted By:
Anik Kumar Chakrabortty
ID: 2017000000037 – Semester: Summer 2018

Solution to Problem 1:

a) Finding the powerful array: C++ Implementation

```
int a[10], b[10]; // given array.
    int cntOne=0, cntTwo = 0;
    for(int i=0; i<10; i++){
        if(a[i]>b[i])
            cntOne++;
        else if(a[i]<b[i])
            cntTwo++;
    }
    if(cntOne>cntTwo)
        cout << "Array A is more powerful" << endl;
    else if(cntTwo>cntOne)
        cout << "Array B is more powerful" << endl;
    else
        cout << "It is a tie!" << endl;
```

b) Finding which one has higher average: C++ Implementation

```
int a[10], b[10];                    // given array
int sumOne=0;
int sumTwo=0;
for(int i=0; i<10; i++){
        sumOne+=a[i];
        sumTwo+=b[i];
}
//Finding average
double avgA = sumOne/10;
double avgB = sumTwo/10;

if(avgA>avgB)
        cout << "Array A has higher average" << endl;
else if(avgB>avgA)
        cout << "Array B has higher average" << endl;
else
        cout << "It is a tie!" << endl;
```

Solution to Problem 2:

a) Which algorithm will work better on a data set that contains 100,000 :

Insertion sort is like sorting a hand of cards. You first sort first 2 cards. Then place the 3rd card in its appropriate position in the 2 sorted cards. Then 4th card is added to sorted 3 cards in the correct position and so on. In Bubble sort, you swap adjacent elements which are out of sorted order. So, the largest element is moved to the end of the array in the first iteration. In the next iteration, the same process is repeated till n-2 elements, so that second largest element is moved to the second last element position. Proceeding this way for n iterations, the array gets sorted. So it is obvious, insertion sort will require a lot less iteration over a given data set. So, on an array of 100,000 elements insertion sort will work better.

b) Procedure to sort the new array created from the old array only taking the elements in even position:

There's a given array let's say it is: int A[20];

Now we need to create a new array of 10 indices which will only hold even positioned data given on previous array.

Previous Array had: 0, 1, 2, 3, 4,……. ,19.

New array B[10] will consist of 1, 3, 5, 7,…..19.

We will use a formula on the iteration so we can find even position. If the iteration is on i'th index, we gonna store i*2+1th index on the new array.

```
for(int i=0; i<10; i++){
        B[i] = A[i*2+1];
```

Now that we have the new array with only 10 indexes retrieved from even position (2, 4, 6…) [not index]

We will use a sorting algorithm to sort this list. As the dataset is very small. Just applying Bubble Sort will give us the desired result.

```
for(int i=0; i<=n-2; i++){
        for(int j=0; j<=n-2; j++){
                swap(a[j+1], a[j]);
        }
}
```

Solution to Problem 3:

a) Which searching technique is better on 1000 numbers given the array elements are in no specific order:

For 1000 Numbers between linear search and binary search technique, Binary Search will work efficiently. But the pre requirement to use binary search on a dataset is that, it needs to be sorted. As the array elements are in no specific order, we are forced to use linear search. And in modern machines, linear search will perform quite fast on 1000 numbers.

b) Implementation of binary search on given list when we need to search x=7.

```
 int binary_search(int arr[], int l, int r, int x){
    int mid = (l+r)/2;
    if (arr[mid] == x){
        return mid;
    }else if (arr[mid] > x){
        binary_search(arr, l, mid - 1, x);
    }else if (arr[mid] < x){
        binary_search(arr, mid + 1, r, x);
   }else{
    return -1;
    }

int main(){
int x=7;
int arr[] = {21, 15,12,10,8,7,6,4,2};
int findIndex = binary_search(arr, 0, 9, x);
if(findIndex==-1)
        cout << “Element not found” <<endl;
else
        cout << “Element found on index:“ << findIndex << endl;
```

Solution to Problem 4:

a) Which data structure to use:

As the number of students is not fixed, it is suitable to use LinkedList on this problem. It is possible to point a new node on a LinkedList implementation easily. Without allocating memory previously, this data structure allows us to store, insert or delete an entry efficiently.

b) Creating a LinkedList for 10 nodes and Checking if a particular value is present or not:

```
 struct Node
{       int data;
        struct Node *next;
};
int main(){
        Node *head, *temp, *x, *y;
        head=new Node;
        cin >> head->data;
        head->next = NULL;
        x=head;
        for(int i=0; i<9; i++){
                temp = new Node;
                cin >> temp->data;
                temp->next = NULL;
                x->next = temp;
                x=x->next;
        }
        y=head; // storing head to y for iterating
        int query; cin >> query; //finding if query exist in list.
        for(int i=0; i<10; i++){
                if( y→data==query)
                        cout << “Found the data on list” << endl;
                y=y->next;
        }
}
```