DATA STRUCTURING CASE STUDIES

Case Study One (1)

Finding the Maximum and Minimum of n integers

How would we approach this task?

First; from our previous knowledge; the following are independent codes for obtaining maximum and minimum separately.

```
int max=numbers[0];
for(int i=1; i<n; i++){
      if (max<numbers[i])
            max=numbers[i];
}
```

Quiz

i. What difference does it make if we assigned the max initially to zero & not the first element in the array?

ii. Would it make a difference if our if statement looked like this:
`if (max<numbers[i])` ?

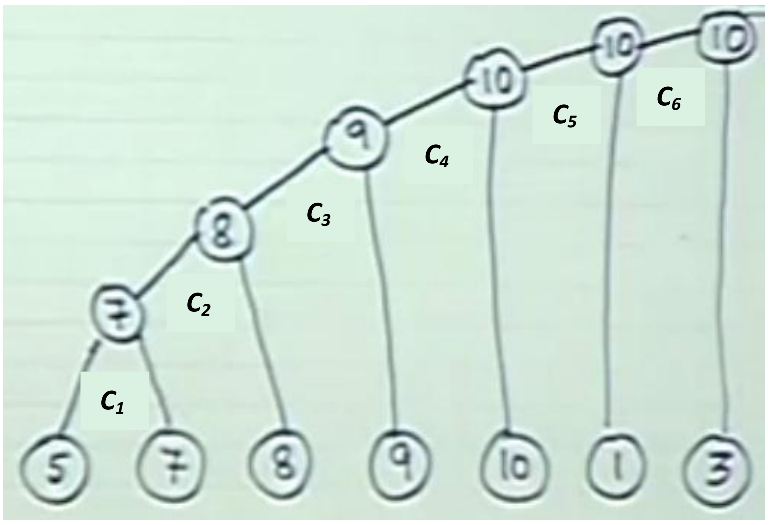
```
int min=numbers[0];
for(int i=1; i<n; i++){
      if (min>numbers[i])
            min=numbers[i];
}
```

Quiz

i. What difference does it make if we initialized the counter i to 0 in the for loop?

ii. Why would we need no else statement in this case?

Diagrammatically, both of these seem to present comparison in the following format

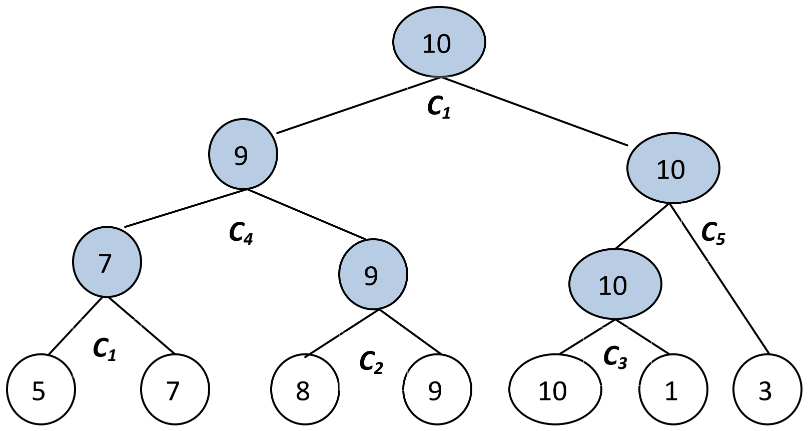


We call this **Approach A**

From this diagram, it is eminent that we need to make n-1 comparisons to obtain the maximum of n integers.

i.e. out of an n=7 we need 6 comparisons.

Question: is there another approach to obtaining this?



Approach B

Suppose we decide to structure our approach differently; such that we sort of group our integers in some way first: in this case in pairs, do we still obtain the maximum? (Yes)

Question: How many comparisons do we need in this approach? In this approach we find that we still need the n-1 number of comparisons.

Observations

From the above, we now make one important observation: that it is sufficient and necessary to make n-1 comparisons in order to obtain the maximum of n numbers.

If we view this structure as a tournament then we have two possible ways of setting up our tournament of n players: in **Approach A:** Depending on the position of the maximum/ winner at the start of the tournament, the winner might need to play all other players, unlike in **tournament B** in which the winner plays only a fraction of the rest: How many players in tournament B does the winner have to play?

Albeit the difference, we still need n-1 comparisons to be able to obtain the winner in a tournament of n players: real world example include games tournaments in which, like the world cup, the tournament structure is in a ways that each group has 4 teams…. Still we must perform n-1 matches to obtain the winner (E.G. FIFA world there are normally 32 teams requiring 31 matches to obtain the winner)

Question: Can we find both the maximum and Minimum of n numbers?

We could achieve this by combining the code for obtaining the maximum with that for obtaining the minimum: The following two pieces of code perform this task, observe them carefully:

```
int max=numbers[0];
int min=numbers[0];

for(int i=1; i<n; i++){
      if (max<numbers[i])
            max=numbers[i];

      if (min>numbers[i])
            min=numbers[i];
}
```

Quiz

Do we actually obtain the maximum and the minimum, if so then how many comparisons do we require?

```
int min=numbers[0];
for(int i=1; i<n; i++){
      if (min>numbers[i])
            min=numbers[i];

      else if (min>numbers[i])
            min=numbers[i];
}
```

Quiz

Answer the same quiz in the left side code.

How different would this code look if we use the approach in quiz ii of the first code for obtaining the maximum?

Observations

i. The code without the else **must,** of necessity, take n-1 comparisons to obtain maximum, and another n-1 comparisons to obtain minimum i.e. $2(n-1)$

ii. The code that includes the else takes a numbers of comparisons that is less then $2(n-1)$; this number c, depends on the actual arrangement of the numbers in the array: even though there are cases when the number of comparisons are actually $2(n-1)$ **(When is this case?).**

iii. **Therefore :**
   a. Without the else: $C = n - 1$
   b. With the else: $C \leq 2(n-1)$

From here it is a question of probability; in which case the probability of the code with an else statement taking 2(n-1) is very low that in the real world scenarios might never exist.

Suppose each comparison takes x amount of time to do, then we are saying that the else statement eliminates some amount of time in our solution.

Conclusion

i. From the foregoing discussions, we realize that a simple else statement can change tremendously the time taken to solve a given problem.

ii. It is possible to have more than one approach to solving a single problem, all of which will give the correct answer.

iii. This is the heart of problem solving; that we no longer just require to write some code that can run and give a correct answer, but we need to ask ourselves questions like:
   a. Are there other alternative approaches?
   b. Can we redefine one approach to reduce its execution time, memory space and other aspects (what will later become known as: **efficiency**)

With this case study we introduce to the student the essence of data structuring and Algorithms, and as we begin this Semester work, this will be the core question ringing in our minds.

A DEEPER LOOK AT THE MAX - MIN PROBLEM

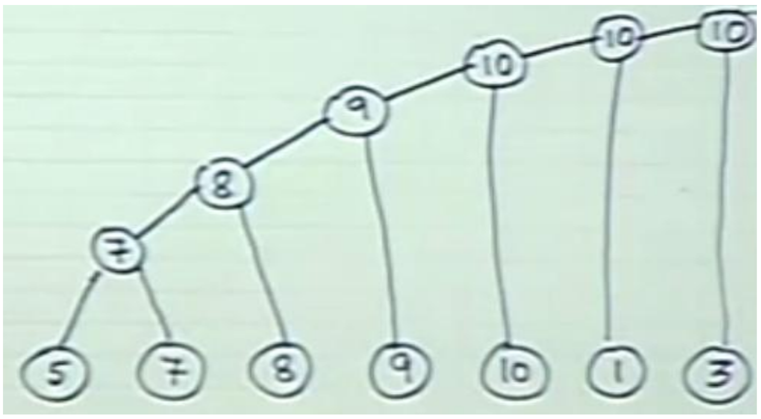
So far our best approach takes $C \leq 2(n-1)$could we further reduce these comparisons so that probably we obtain a C that can never equal $2(n-1)$.

The answer to this question is inherent in a property of the minimum. What do we know about the minimum number or the ultimate looser that could help us?

Logically reasoning; the minimum or the ultimate looser cannot be the maximum or the winner, can it? If we expand this thought, we find that actually the minimum cannot be among any of those players who won at least one of their matches. In a sense, when we have done well to find the maximum using n-1 comparisons; we do not need to compare just any other numbers to obtain the minimum but **we need only those players who lost their matches**, i.e. the ultimate looser must first be a looser: he is a looser of losers.

Our task therefore is to find this **SET OF LOOSERS (L)** and somehow minimize it as much as possible

Another look at approaches A and B



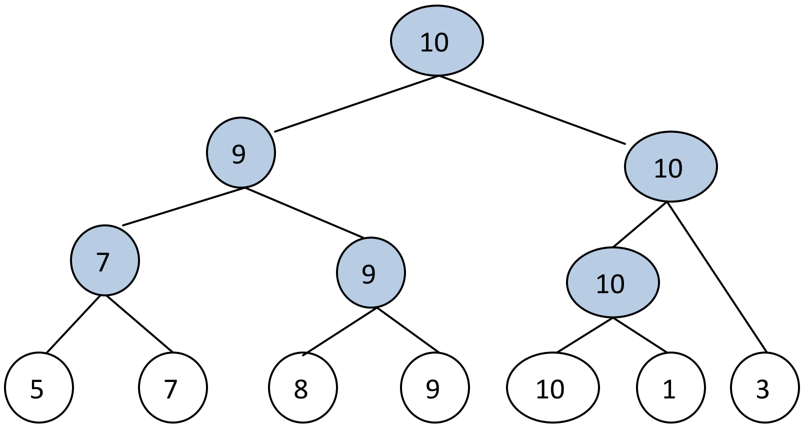
Approach A

SET $\boldsymbol{L} = \{\mathbf{5}, \mathbf{7}, \mathbf{8}, \ \mathbf{9}, \mathbf{1}, \mathbf{3}\}$

If we take the numbers that lost in the comparisons we have $n - 1$ elements.

If the largest number was the first element in the list, then all the others must lose to it (mandatory n-1)

We still need $2(n - 1)$ comparisons



Approach B

SET $\boldsymbol{L} = \{\mathbf{5}, \mathbf{8}, \mathbf{1}, \mathbf{3}\}$

Here we have fewer elements: By setting the tournament in pair wise structure we find that only half of the elements form the set of losers. $(\frac{n}{2})$

To obtain max and min then; we require $(n - 1)$ for maximum and $(\frac{n}{2} - 1)$ for minimum

This gives us $(\frac{3n}{2} - 2)$ comparisons

Observations

i. No. of comparisons to find MAX=N-1
ii. Minimum can be obtained from those numbers (players) that lost in their first comparisons in the tournament.
iii. This set of losers must be minimized to make the algorithm **efficient.** We observe that this set will be least if pair wise comparison is done.
iv. There are two possible structures A and B**, Approach B:** becomes better in this sense: only if you want to obtain both MAX and MIN. otherwise if you wanted only the max then both algorithms can do.

Conclusion

The manner in which we structure our data determines how much time we spend processing it. We then need to have two lists during the first pair wise comparison; A list W of winners and a list L of losers, anyone who loses goes to L and anyone who wins goes to W. then we obtain a winner by comparing the winners and a loser by comparing the losers.

SOME CODE

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

/*A function to get maximum and minimum
  Numbers is the array of numbers, n is the size of the array i.e.  number of numbers in the array
*/
void max_min(int[] numbers, int n){
      int p=0, k=0, max, min, i;
      int winners[n/2], losers[n/2];

      /*carry out pair wise comparison placing the larger numbers in the set winners and the smaller ones in the set losers*/
      for(int i=0; i<n; i+2){
            if(numbers[i]<numbers[i+1]){
                  losers[p]=numbers[i];
                  winners[k]=numbers[i+1];
            }
            else{
                  losers[p]=numbers[i+1];
                  winners[k]=numbers[i];
            }
      }

      //output the results by calling the functions for obtaining max and min
      cout<<"Maximum = "<<getMaximum(winners,n/2);
      cout<<"Minimum = "<<getMinimum(loosers,n/2);
}

//A function to obtain the maximum of n numbers given an array (called large of size n)
int getMaximum(int[] large, int n){
      int max=large[0];

      for(int i=0; i<n; i++){
            if(max<large[i])
                  max=large[i];
      }

      return max;
}
```

```
//A function to obtain the minimum of m numbers given an array (called small of size m)
int getMinimum(int[] small, int m ){
      int min=small[0];

      for(int i=0; i<n; i++){
            if(min<small[i])
                  min=large[i];
      }

      return min;
}


/*The main Function; this function asks the user to enter the numbers he wants,
   Then it calls the max-min function passing the array to it*/

void main(){
   int n;
   cout<<"\n THIS IS A PROGRAM THAT OBTAINS BOTH MAXIMUM AND MINIMUM"<<endl;
   cout<<"_____________________________________________________"<<endl;
   cout<<"\n Enter the number of numbers n : "<<endl;
   cin>>n;

   int the_array[n]; //an array to hold the n integers

   for(int i=0; i<n; i++){
      cout<<" Enter a number ["<<i<<"of"<<n<<"] Already entered"<<endl;
      cin>> the_array[i];
   }
}
```

A Note to The Student
There are several concepts implemented in this piece of code; study this and be apple to appreciate these concepts namely:

1. Functions: how and where they are defined.
2. Parameters: Formal and actual parameters, parameter passing and how to pass arrays as parameters.
3. Iteration and how to take advantage of its power.

Question
The code declares and uses two other arrays of sizes n/2; This means that we require twice as much space as our data. Can we reduce this? I.E. is it possible to create these two sets Losers and Winners without involving two different arrays (Can we do it within the same array)? If we can get an answer to this then we will save on space.