PAGE#1

Aisha Muhammad Nawaz
20L-0921
BSCS-4A1
Design And Analysis
of Algorithms Spring 2022
Dr. Maryam Bashir
Home Work # 3 Due : 20-03-22 Sunday

Question# 1

(a). 1. c++ code
2. c++ code
3. c++ code
4.

* my code was giving exception with array of size 10^8 so I used an array of size 10^6.

#	Merge Sort No. of Comparisons	Min Heap Sort No. of Comparisons	Quick sort No. of comparisons
1	18673677	38894637	25945467
2	18674800	38895793	25122794
3	18674490	38891369	25116567
4	18673957	38890908	25894573
5	18673744	38892994	24621012
Average	18674133.6	38893140.2	25340082.6

(b).

#	Merge Sort time in millisecond	Heap Sort time in millisecond	Quick Sort time in Millisecond
1	2291	2809	1144
2	2147	2857	1110
3	2173	2753	1133
4	2167	2788	1137
5	2321	2870	1121
AVG	2219.8	2815.4	1129

PAGE#2

(c).- The Algorithm of ~~Insertion~~ Intro Sort finds the need to switch ~~to~~ from Quick sort because Quick sort's worst case is O(N²) and needs to be avoided. So when there is a chance of worst case it switches to from Quick sort which also avoids increasing the recursion stack space O(logN)). The switch is made to Heap Sort. Because of large constant factor, quicksort can ~~perfor~~ so even worse than O(N2) when N is small enough. So a switch is made to insertion sort to decrease running time.

→ If a bad pivot is selected quicksort does little good.

Name: Aisha Muhammad Nawaz
Roll# 20L-0921
Section: BSCS 4A
Course: Design And Analysis of Algorithms
Instructor: Dr. Maryam Bashir
Homework # 3 (Due: 15-03-22)
Tuesday 10am

Question # 2 ALGORITHM

1) We will use hash map for this solution. First store all possible binary representations of numbers from 0 to n-1 in hashmap. Add /Insert $O(1)$ them along with a bool value initialized to false. So basically the hashmap will contain key-value pairs where key will be the string type binary representations and value will be a flag of type bool set to false initially.

2) Using a for loop all original binary representations till n-1 will be inserted into hashmap along with bool type value set to false

3) Moving on, another loop below the above mentioned will separately work. The function of this loop will be to

(0 → n-1) one by one find corresponding values from our source (where we are checking missing value to be in) with our hashmap. If that value is found in hashmap then change its value to true. O(n)

4) Below, Another final loop will work separately. This loop will check to find if any key has value false if so. it will save the value in a temporary string and break out of loop O(n)

Question 3 →

```
bool Find Matching (int* Arr, int l, int r, int &i)
{
        if( l < r )
        {
            int  m= (l+r)/2;
            Find Matching (Arr, l, m, i);
            Find Matching (Arr, m+1, r, i );
        }
        else
        {   i++;
            if  (Arr[l] == i)
            {
               cout << "Match found for: " << Arr[l];
               cout << " i: " << i << endl;
               return true;
            }
            return false;
        }
};
```

*Here Arr is the array that contain the values. 'l' is left index of array (intially zero), 'r' is right index of array (intially for eg. 4 for an array of size 5) and i is an iterator that keeps track of index and is intially -1.

```
int  main ()
{
     int arri [ ]  = {4, 5, 2, 8};
     int x = -1;
     int &i = x;
      Find matching (arri, 0, 3, i);

  return 0;
}
```

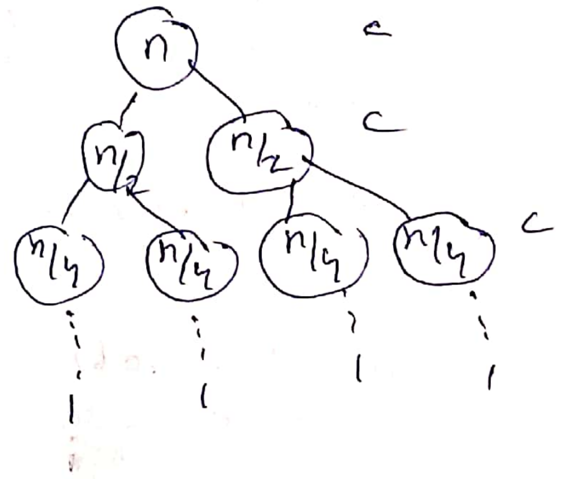
Output will be:

```
Match found for : 2
i : 2
```

My Algorithm runs in O(logn) time because each time the array is being divided into two equal parts and at every level constant amount of work is done. Total levels are $\log_2 n$

So O(logn).

$T(n) = T(n/2) + T(n/2) + c$



Total number of levels: $\longrightarrow \frac{n}{(2)^k} = 1$

$n = 2^k$

$\log_2 n = K$

Total workdone $\longrightarrow$ $c \cdot \log_2 n$

So $\longrightarrow$ O(logn)

Question 4 $\longrightarrow$

```
bool   Find Position (int* Arr, int l, int r, int &i, int x)
{          if ( isinf (Arr[L])) //infinity reached
                 return false;
           if (L<r)
      {    int   m = (L+r)/2;
           Find Position (Arr, L, m, i, x);
           Find Position (Arr, L, m+1, r, i, x);

      }
   else
      {     i++;
            if ( Arr[i] == x)
            { cout<<" The position is : "<< i<<endl;
            return true;

            }
            else
               return false;

      }
  };
```

* Here Arr is the array that contains all the values. 'l' is left index of array (initially zero), 'r' is right index of array (initially for eg.4 for an array of size 5) and i is an iterator that keeps track of position/ index and is initially -1. x is the integer to be found.

```
int   main ()
{  int   arri[]= {4,5,2,∞};
      int L = -1;
      int &i = L;
        Find position (arri,0,3, i, 5);
           return 0;
}
```

output will be:
The Position is : 1

Question 5 →

```
int   Find Distance  (A, l, r, x, y)
{
          if (l==r)
                return  array[l]
          m= (l+r)/2

          int   d  =0;
                    Find Distance (A, l, r, x, y)
                    Find Distance. (A, m+l, r, x, y)
}             d= Find x Y Distane (A, l, m, r, x, y) ;
             return d;


int   Find   XY Distance ( A, l ; m, r , x, y);
   {        int dist= 0;
            for (i=m to l)
                {
                        if (A[i] ==x || A[i]==y)
                              break;
                        else
                              dist ++;
                }
            for (i= m+1 to r)
                  {  if (A[i] ==y || A[i]==x)
                              break ;
                        else dist++;
                  }
            return dist ;
   }
```

* This is Divide and Conquer Solution

O (n lgn)