Name: Fadi Alahmad Alomar

ID: 120180049

Assignment 4: Divide and Conquer

Question 1

1. $T(n) = 5T(\frac{n}{2}) + \mathcal{O}(n)$
   Using the master theorem we get:
   $b = 2,\ a = 5,\ f(n) = n$
   $c_{crit} = \log_b a = \log_2 5 = 2.322$
   $f(n) = n = \mathcal{O}(n^c)$ where $c = 1$
   $\because c_{crit} > c$
   $\therefore T(n) = \Theta(n^{c_{crit}}) = \Theta(n^{2.322})$

2. $T(n) = 2T(n-1) + \mathcal{O}(1)$
   $T(n-1) = 2T(n-2) + \mathcal{O}(1)$
   $\therefore T(n) = 2 * (2T(n-2) + \mathcal{O}(1)) + \mathcal{O}(1)$
   $T(n) = 2^2 * T(n-2) + 2\mathcal{O}(1)$
   In general:
   $T(n) = 2^k * T(n-k) + 2^{k-1}\mathcal{O}(1) = 2^k * T(n-k) + \mathcal{O}(2^{k-1})$
   for $T(1)$ then $k = n-1$
   $T(n) = 2^{n-1} * T(1) + \mathcal{O}(2^{n-2}) = 2^{n-1} + \mathcal{O}(2^{n-2}) = \mathcal{O}(2^n)$

3. $T(n) = 9T(\frac{n}{3}) + \mathcal{O}(n^2)$
   Using the master theorem we get:
   $b = 3,\ a = 9,\ f(n) = n^2$
   $c_{crit} = \log_b a = \log_3 9 = 2$
   $f(n) = n = \mathcal{O}(n^c \log^k n)$ where $c = 2, k = 0$
   $\because c = c_{crit}$
   $\therefore T(n) = \Theta(n^c \log^{k+1} n) = \Theta(n^2 \log n)$

The best algorithm is algorithm C with running time $\Theta(n^2 \log n)$

Question 2

$T(n) = 2T(\frac{n}{2}) + \mathcal{O}(1)$
Using the master theorem we get:
$b = 2,\ a = 2,\ f(n) = 1$
$c_{crit} = \log_b a = \log_2 2 = 1$
$f(n) = 1 = \mathcal{O}(n^c)$ where $c = 0$
$\because c_{crit} > c$
$\therefore T(n) = \Theta(n^{c_{crit}}) = \Theta(n)$

Question 3

By doing a reverse Binary search.
we start at the first element and check if it is infinity then we double our index and check again until we reach infinity and once we reach it it is know that the index we are at is at mx $2 * n$ if the iteration before last landed on the last element then we do a normal binary search in the portion of the array from the start to the index we found.

```
index -> 1                  # assume our indexing starts from 1
while arr[index] != INF:
    index -> 2*index
BinarySearch(arr[:index],x) # binary search for x  in the array from
                            #  the start to the index we found
```

this gives the algorithm a run time of:
$T(n) = \mathcal{O}(\log 2 * n) + \mathcal{O}(\log 2 * n) = \mathcal{O}(\log n)$

Question 4

We binary search the array but instead of checking for a value if bigger or smaller we check $A[i]$ and i

```
binarySearch(arr, size):
    beg -> 0
    end -> size
    while beg!=end
        index = (beg + end)/2
        if (index == arr[index])
            return True
        else if (index > arr[index])
            beg = index + 1
        else
            end = index - 1
    return False
```

Question 5

Using decision tree methodology.
At each node a comparison is made A[i] ≤ x which decides what happens next in this tree there are $n+1$ leaf nodes as there are n possible places for x to be and 1 is if it is not presented in the array, thus the shortest path from the route to a leaf node is the height which is $\log n + 1$
$\therefore$ any comparison search must take do least $\log n + 1$ comparisons $\therefore \Omega(\log n)$

Question 6

By going the array to find min and max, then creating a new auxiliary array that has the size max-min, then going through the array again and putting each element in the new array using the element - min as index, then creating a new answer array of the same size as the original and going through the auxiliary array and putting the elements in the answer array as they appear.

```
Sort(arr)
    mn -> inf
    mx -> -inf
    for int i = 0; i< arr.size; i-> i+1:
        if arr[i]>mx:
            mx -> arr[i]
        if arr[i]<mn:
            mn -> arr
    M -> mx - mn
    auxArr -> Array[M+1]
    for int i = 0; i< arr.size; i-> i+1:
        auxArr[arr[i]-mn] -> arr[i]
    ansArr -> Array[arr.size]
    j -> 0
    for int i = 0; i< auxArr.size; i-> i+1:
        if auxArr[i] is not NULL:
            ansArr[j] -> auxArr[i]
            j->j+1
    return ansArr
```

the $\Omega(n \log n)$ bound does not hold in this case as this algorithm is not based on comparisons.