EL9343 Homework 03

Spring 2022

Name: Sagar Patel
NETID: SP5894

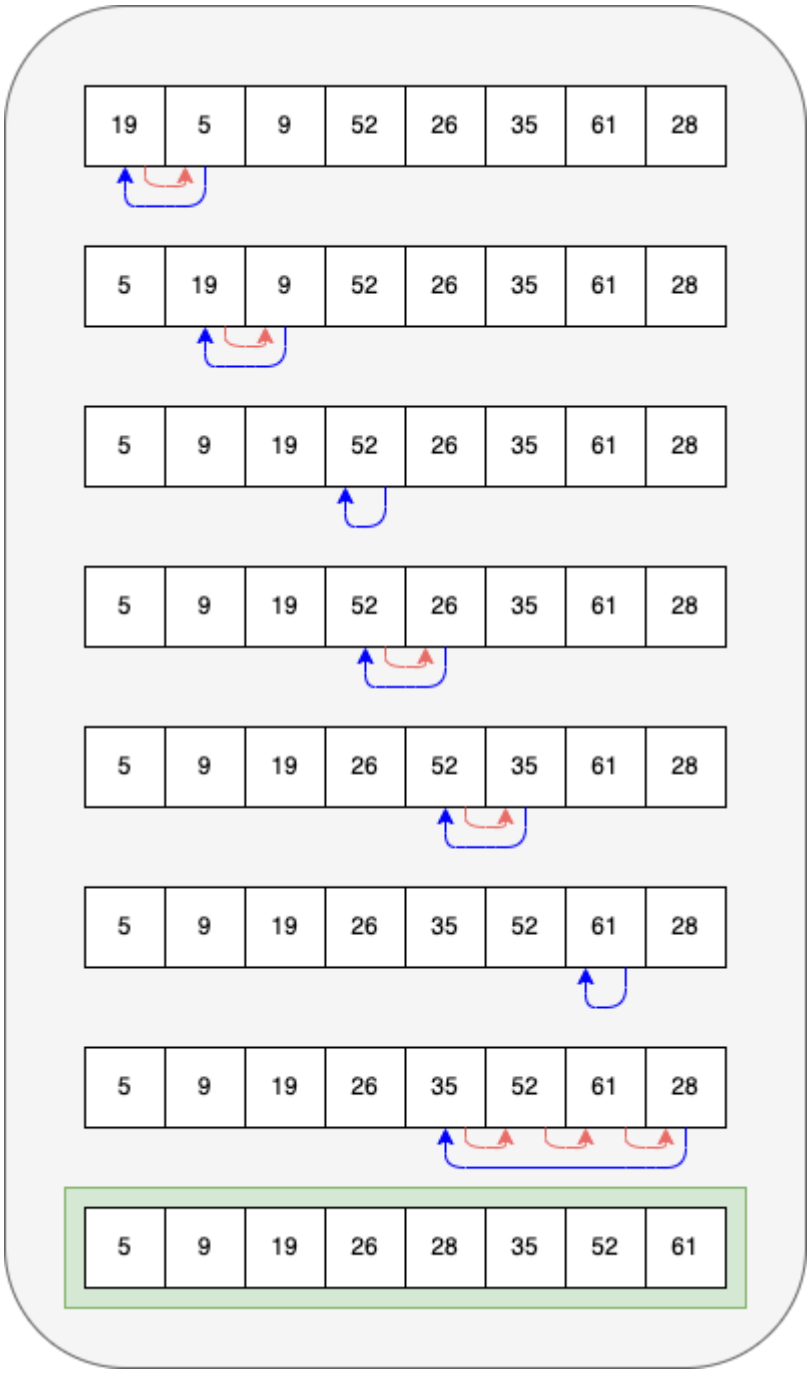
Question 01

Given the array $A[19, 5, 9, 52, 26, 35, 61, 28]$

1.a

Using Figure 2.2 on page 18 of CLRS as a model, illustrate the operation of insertion-sort on A.

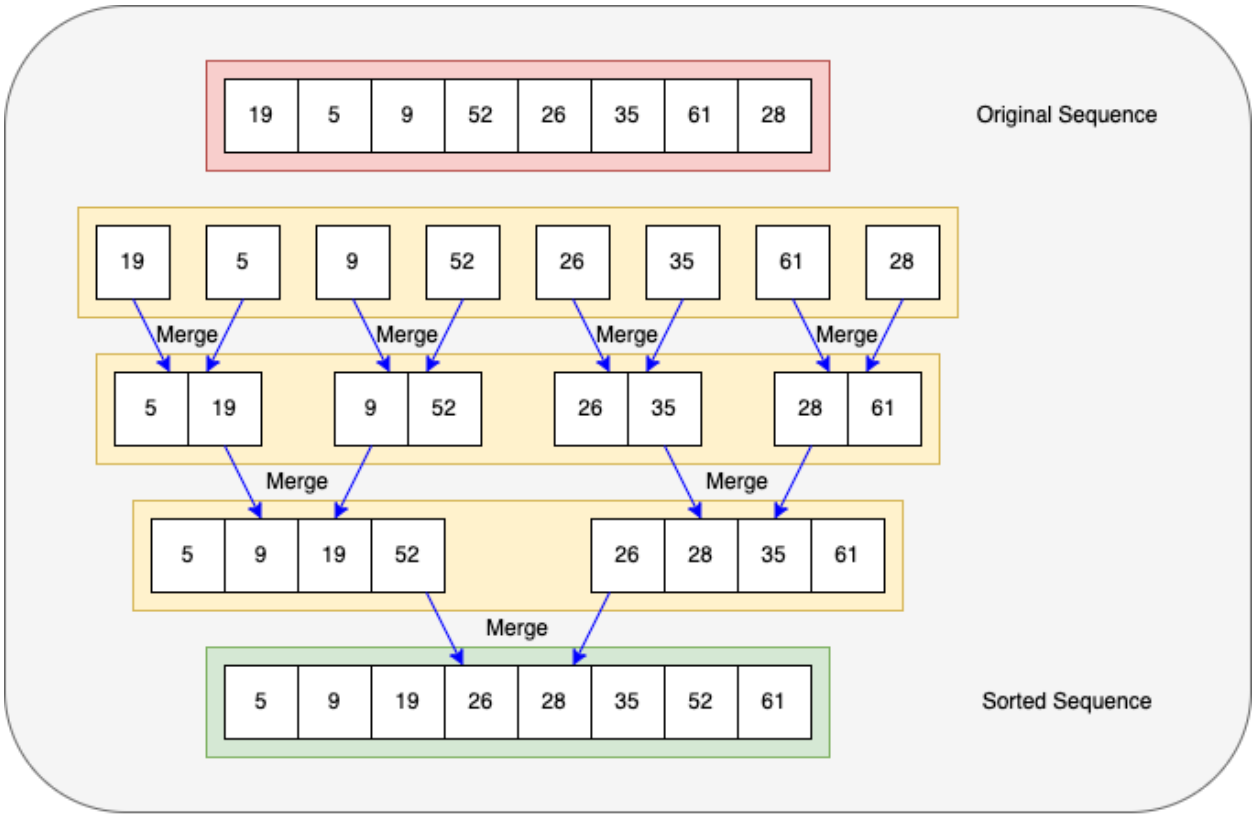
Solution



1.b

Using Figure 2.4 on page 35 of CLRS as a model, illustrate the operation of merge-sort on A.

Solution



Question 02

Consider sorting n numbers stored in array A by first finding the smallest element of A and exchanging it with the element in $A[1]$. Then find the second smallest element of A, and exchange it with $A[2]$. Continue in this manner for the first $n-1$ elements of A.

2.a

Write pseudocode for this algorithm, which is known as $selection\ sort$.

Solution

```
n = length(A)
for j=1 to (n-1):
    smallestValue=j
    for i=j+1 to n:
        if A[i]<A[smallestValue]:
            smallestValue = i

    swap A[j] with A[smallestValue]
```

2.b

What loop invariant does this algorithm maintain?

Solution

At the start of each iteration of the outer `for` loop, the subarray $A[1, \ldots, j-1]$ consists of the smallest $(j-1)$ elements in array $A[1, \ldots, n]$, and that subarray is then in the sorted order.

2.c

Why does it need to run for only the first $n-1$ elements, rather than for all n elements?

Solution

After the first $n-1$ elements, the subarray $A[1, \ldots, n-1]$ contains the smallest $(n-1)$ elements, sorted, and therefore element $A[n]$ must be the largest element.

2.d

Give the best-case and worst-case running times of selection sort in Θ-notation.

Solution

The running time of the algorithm is $\Theta(n^2)$ for all cases.

Question 03

For the maximum subarray problem, if we use divide-conquer, but instead of dividing the array into two halves, we equally divide it into three segments, how should the algorithm be modified? What is the running time of the new algorithm?
Note: Pseudocode is not necessary. You can explain your algorithm and running time step by step.

Solution

If we choose to divide the original array A into three equally-sized subarrays S_1, S_2, and S_3 and then we have three scenarios to consider in the combined phased of the *divide-and-conquer* algorithm.

--- Scenario A: The maximum subarray starts from S_1 and ends in S_2
A Linear scan is performed in S_1 from tail to head and a scan is performed in S_2 from head to tail.
The sum is then given by $\Rightarrow \Theta(\frac{2n}{3}) + \Theta(1)$

--- Scenario B: The maximum subarray starts from S_2 and ends in S_3
A Linear scan is performed in S_2 from tail to head and a scan is performed in S_3 from head to tail.
The sum is then given by $\Rightarrow \Theta(\frac{2n}{3}) + \Theta(1)$

--- Scenario C: The maximum subarray starts from S_1 and ends in S_3
A Linear scan is performed in S_1 from tail to head and a scan is performed in S_3 from head to tail.

Additionally, we also need to computer the sum of S_2 for summation $\Rightarrow \Theta(n) + \Theta(1)$

We also need to recursively find the maximum subarray in S_1, S_2, and S_3 and so we can use the recurrence formula to calculate that −
$T(n) = 3T(\frac{n}{3}) + \Theta(\frac{2n}{3}) + \Theta(1) + \Theta(\frac{2n}{3}) + \Theta(1) + \Theta(n) + \Theta(1) \Rightarrow \Theta(n) + 3T(\frac{n}{3})$
$\therefore T(n) = \Theta(n.logn)$

Question 04

The Tower of Hanoi is a mathematical game or puzzle consisting of three rods and a number of disks of various diameters, which can slide onto any rod. The puzzle begins with n disks stacked on a start rod in order of decreasing size, the smallest at the top, thus approximating a conical shape. The objective of the puzzle is to move the entire stack to the end rod, obeying the following rules:

1. Only one disk may be moved at a time.
2. Each move consists of taking the top disk from one of the rods and placing it on top of another rod or on an empty rod.
3. No disk may be placed on top of a disk that is smaller than it.
   To print each move of this game with the minimum number of steps, please fill in the pseudo code in the MOVE function.
   For example, ideal output format should be like this：

```
>>> MOVE(3, 1, 3)
Move the top disk from rod 1 to rod 3
Move the top disk from rod 1 to rod 2
Move the top disk from rod 3 to rod 2
Move the top disk from rod 1 to rod 3
Move the top disk from rod 2 to rod 1
Move the top disk from rod 2 to rod 3
Move the top disk from rod 1 to rod 3
```

Solution

```
PRINT(origin, destination):
print("Move the top disk from rod ", origin, "to rod ", destinatio
n)

MOVE(n, start, end):
middle = 6 - start - end                    //Middle rod stands for t
he remaining rod
if n==1:
    print(start, end)
else:
    MOVE(n-1, start, middle)
    MOVE(1, start, end)
    MOVE(n-1, middle, end)
```

Question 05

For an array with n elements, design a *divide-and-conquer* algorithm for finding both the minimum and the maximum element of this array using no more than $3n/2$ comparisons. (Pseudocode is required)

Solution

Whenever we want to get the maximum or the minimum of an array, say A, conventionally, we can divide it into two parts and compute the maximum and minimum respectively. We can then combine them to get the maximum and minimum of the array A. In order to understand it better, the pseudocode is as shown −

```
struct miniMaxi
{
    minimum
    maximum
    miniMaxi(A, B)
        minimum = A
        maximum = B
};

minMax(A, low, high)
    if (low==high):        //When there is only one item in A
        return miniMaxi(A[low], A[low])
    if (high == low+1):    //When there are two items in A
        if (A[low] > A[high]):
            return miniMaxi(A[high], A[low])
        else:
            return miniMaxi(A[low], A[high])

    middle = (low + high)/2
    left = minMax(A, low, middle)
    right = minMax(A, middle+1, high)

    if (left.minimum < right.minimum):
        tmp1 = left.minimum
    else:
        tmp1 = right.minimum

    if (left.maximum < right.maximum):
        tmp2 = right.maximum
    else:
        tmp2 = left.maximum

    return miniMaxi(tmp1, tmp2)
```

--- Analysing the running time

There seem to be two recursive calls and all the other operations can be done in $O(1)$, we can have the recursive formula to be $\Rightarrow T(n) = 2T(\frac{n}{2}) + \Theta(1)$

Using the master method, we have $T(n) = \Theta(n)$ which is in the linear time.

If n is a power of 2, we have $\Rightarrow T(n) = (\frac{3n}{2} - 2) \leq \frac{3n}{2}$

Therefore, it is safe to say that the divide and conquer algorithm uses no more than $\frac{3n}{2}$ comparisons.