**Divide & Conquer**

Many algorithms are recursive in nature to solve a given problem recursively dealing with sub-problems.

In divide and conquer approach, a problem is divided into smaller problems, then the smaller problems are solved independently, and finally the solutions of smaller problems are combined into a solution for the large problem.

Generally, divide-and-conquer algorithms have three parts −

**• Divide** the problem into a number of sub-problems that are smaller instances of the same problem.

**• Conquer** the sub-problems by solving them recursively. If they are small enough, solve the sub-problems as base cases.

**• Combine** the solutions to the sub-problems into the solution for the original problem.

**Pros and cons of Divide and Conquer Approach**

Divide and conquer approach supports parallelism as sub-problems are independent. Hence, an algorithm, which is designed using this technique, can run on the multiprocessor system or in different machines simultaneously.

In this approach, most of the algorithms are designed using recursion, hence memory management is very high. For recursive function stack is used, where function state needs to be stored.

Application of Divide and Conquer Approach

Following are some problems, which are solved using divide and conquer approach.

• Finding the maximum and minimum of a sequence of numbers

• Strassen’s matrix multiplication

• Merge sort

• Binary search

**Binary Search**

In this chapter, we will discuss another algorithm based on divide and conquer method.

**Problem Statement**

Binary search can be performed on a sorted array. In this approach, the index of an element x is determined if the element belongs to the list of elements. If the array is unsorted, linear search is used to determine the position.

**Solution**

In this algorithm, we want to find whether element x belongs to a set of numbers stored in an array numbers[]. Where l and r represent the left and right index of a sub-array in which searching operation should be performed.

**Algorithm:** Binary-Search (numbers[ ], x, l, r)

if l = r then

return l

else

m := ⌊(l + r) / 2⌋

if x ≤ numbers[m] then

return Binary-Search (numbers[ ], x, l, m)

else

return Binary-Search (numbers[ ], x, m+1, r)

Analysis

Linear search runs in O(n) time. Whereas binary search produces the result in O(log n) time

**Example:**



**Max-Min Problem:**

Let us consider a simple problem that can be solved by divide and conquer technique.

**Problem Statement**

The Max-Min Problem in algorithm analysis is finding the maximum and minimum value in an array.

**Solution**

To find the maximum and minimum numbers in a given array numbers[ ] of size n, the following algorithm can be used. First we are representing the naive method and then we will present divide and conquer approach.

**Naïve Method**

Naïve method is a basic method to solve any problem. In this method, the maximum and minimum number can be found separately. To find the maximum and minimum numbers, the following straightforward algorithm can be used.

**Algorithm:** Max-Min-Element (numbers[ ])

max := numbers[1]

min := numbers[1]

for i = 2 to n do

if numbers[i] > max then

max := numbers[i]

if numbers[i] < min then

min := numbers[i]

return (max, min)

**Analysis**

The number of comparison in Naive method is 2n - 2.

The number of comparisons can be reduced using the divide and conquer approach. Following is the technique.

**Divide and Conquer Approach**

In this approach, the array is divided into two halves. Then using recursive approach maximum and minimum numbers in each halves are found. Later, return the maximum of two maxima of each half and the minimum of two minima of each half.

In this given problem, the number of elements in an array is y−x+1y−x+1, where y is greater than or equal to x.

Max−Min(x,y)Max−Min(x,y) will return the maximum and minimum values of an array numbers[x...y]numbers[x...y].

**Algorithm:** Max - Min(x, y)

if x – y ≤ 1 then

return (max(numbers[x], numbers[y]), min((numbers[x], numbers[y]))

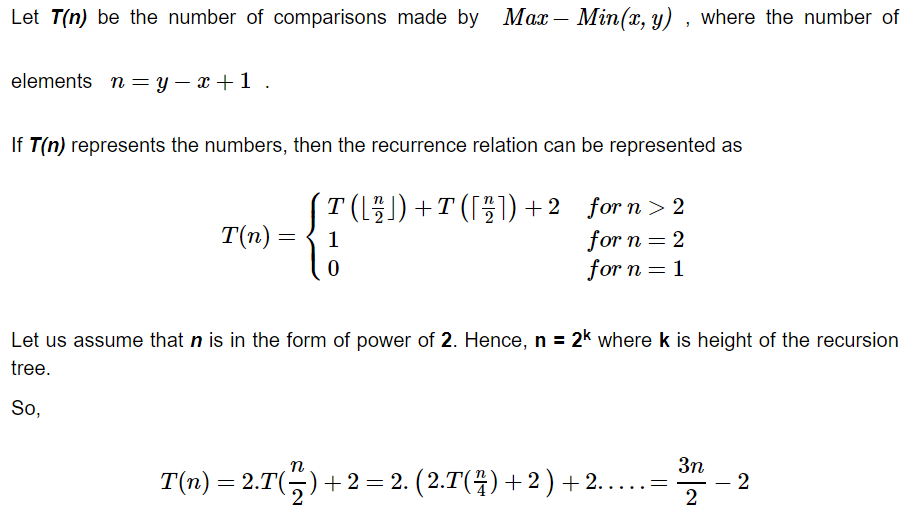
else

(max1, min1):= maxmin(x, ⌊((x + y)/2)⌋)

(max2, min2):= maxmin(⌊((x + y)/2) + 1)⌋,y)

return (max(max1, max2), min(min1, min2))

**Analysis**

****

**RELEVANT READING MATERIAL AND REFERENCES:**

**Source Notes:**

1. <https://www.geeksforgeeks.org/binary-search/>
2. <https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_max_min_problem.htm#:~:text=is%20the%20technique.-,Divide%20and%20Conquer%20Approach,in%20each%20halves%20are%20found.&text=In%20this%20given%20problem%2C%20the,than%20or%20equal%20to%20x.>

**Lecture Video:**

1. <https://www.youtube.com/watch?v=2Rr2tW9zvRg>
2. <https://www.youtube.com/watch?v=uEUXGcc2VXM>
3. <https://www.youtube.com/watch?v=C2apEw9pgtw>

**Online Notes:**

1. <http://vssut.ac.in/lecture_notes/lecture1428551222.pdf>

**Text Book Reading:**

1. Cormen, Leiserson, Rivest, Stein, “*Introduction to Algorithms*”, Prentice Hall of India, 3rd edition 2012. problem, Graph coloring.

**In addition: PPT can be also be given.**