Max-Min Problem

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

Max-Min Problem

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**.

Naive Method

Naive 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)
```

Example

Following are the implementations of the above approach in various programming languages −

C C++ Java Python

```c
#include <stdio.h>
struct Pair {
    int max;
    int min;
};
// Function to find maximum and minimum using the naive algorith
struct Pair maxMinNaive(int arr[], int n) {
    struct Pair result;
    result.max = arr[0];
    result.min = arr[0];
    // Loop through the array to find the maximum and minimum va
    for (int i = 1; i < n; i++) {
        if (arr[i] > result.max) {
            result.max = arr[i]; // Update the maximum value if
        }
        if (arr[i] < result.min) {
            result.min = arr[i]; // Update the minimum value if
        }
    }
    return result; // Return the pair of maximum and minimum val
}
int main() {
    int arr[] = {6, 4, 26, 14, 33, 64, 46};
    int n = sizeof(arr) / sizeof(arr[0]);
    struct Pair result = maxMinNaive(arr, n);
    printf("Maximum element is: %d\n", result.max);
    printf("Minimum element is: %d\n", result.min);
    return 0;
}
```

Output

```
Maximum element is: 64
Minimum element is: 4
```

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 $y - x + 1$, where **y** is greater than or equal to **x**.

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

```
Algorithm: Max - Min(x, y)
if y – x ≤ 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))
```

Example

Following are implementations of the above approach in various programming languages −

C C++ Java Python

```c
#include <stdio.h>
// Structure to store both maximum and minimum elements
struct Pair {
    int max;
    int min;
};
struct Pair maxMinDivideConquer(int arr[], int low, int high) {
    struct Pair result;
    struct Pair left;
    struct Pair right;
    int mid;
    // If only one element in the array
    if (low == high) {
        result.max = arr[low];
        result.min = arr[low];
        return result;
    }
    // If there are two elements in the array
    if (high == low + 1) {
        if (arr[low] < arr[high]) {
            result.min = arr[low];
            result.max = arr[high];
        } else {
            result.min = arr[high];
            result.max = arr[low];
        }
        return result;
    }
    // If there are more than two elements in the array
    mid = (low + high) / 2;
    left = maxMinDivideConquer(arr, low, mid);
    right = maxMinDivideConquer(arr, mid + 1, high);
    // Compare and get the maximum of both parts
    result.max = (left.max > right.max) ? left.max : right.max;
    // Compare and get the minimum of both parts
    result.min = (left.min < right.min) ? left.min : right.min;
    return result;
}
int main() {
    int arr[] = {6, 4, 26, 14, 33, 64, 46};
    int n = sizeof(arr) / sizeof(arr[0]);
    struct Pair result = maxMinDivideConquer(arr, 0, n - 1);
    printf("Maximum element is: %d\n", result.max);
    printf("Minimum element is: %d\n", result.min);
    return 0;
}
```

Output

```
Maximum element is: 64
Minimum element is: 4
```

Analysis

Let **T(n)** be the number of comparisons made by $Max - Min(x, y)$, where the number of elements $n = y - x + 1$.

If **T(n)** represents the numbers, then the recurrence relation can be represented as

$$T(n) = \begin{cases} T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\lceil \frac{n}{2} \rceil\right) + 2 & for\, n > 2 \\ 1 & for\, n = 2 \\ 0 & for\, n = 1 \end{cases}$$

Let us assume that **n** is in the form of power of **2**. Hence, **n = 2^k^** where **k** is height of the recursion tree.

So,

$$T(n) = 2.T(\frac{n}{2}) + 2 = 2.\left(2.T(\frac{n}{4}) + 2\right) + 2 \ldots\ldots = \frac{3n}{2} - 2$$

Compared to Naïve method, in divide and conquer approach, the number of comparisons is less. However, using the asymptotic notation both of the approaches are represented by **O(n)**.