Experiment No. 4
Finding Maximum and Minimum
Name : Vaidehi D. Gadag
Branch/Div.: Comps-1 (C47)
Date of Performance: 07/03/2024
Date of Submission: 14/03/2024

Experiment No. 4

Title: Finding Maximum and Minimum

Aim: To study, implement, analyze Finding Maximum and Minimum Algorithm using Greedy method

Objective: To introduce Greedy based algorithms

Theory:

Maximum and Minimum can be found using a simple naïve method.

1. 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, thefollowing straightforward algorithm can be used.

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

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

2. 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 oftwo 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 orequal to x.

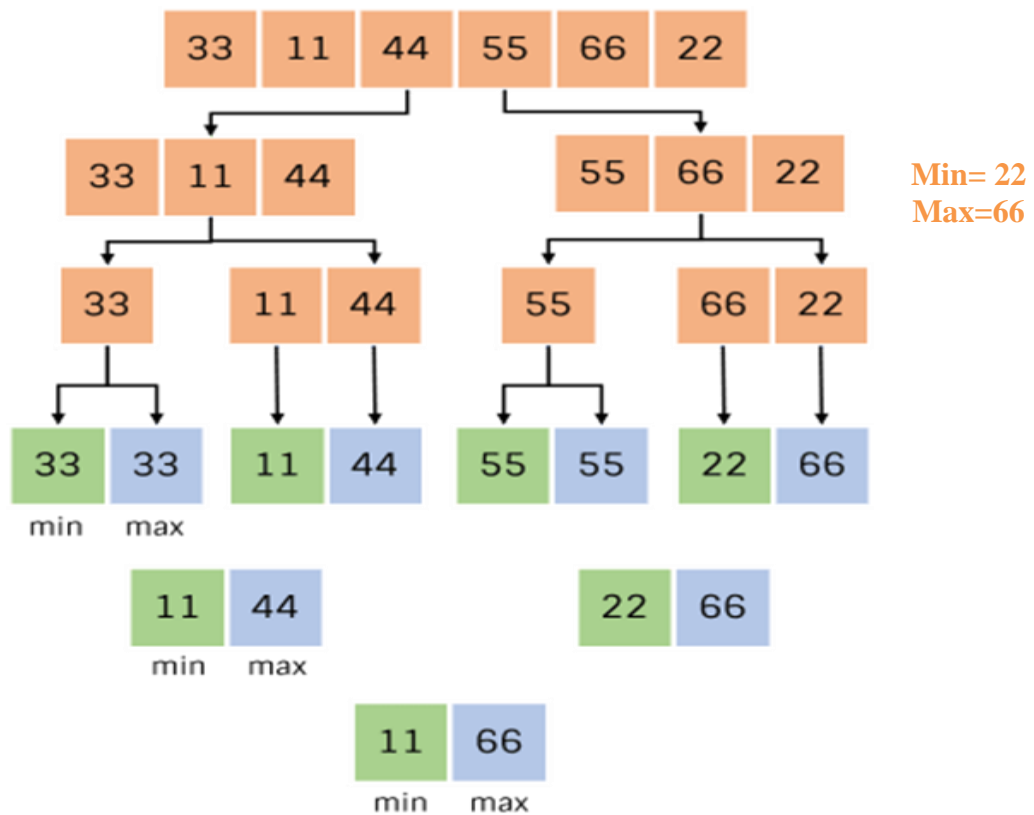DC_MAXMIN (A, low, high) will return the maximum and minimum values of an array numbers[x...y].

Example:

Min= 11
Max=66

Min= 11
Max=44

Min= 22
Max=66

| 33 | 11 | 44 | 55 | 66 | 22 |

| 33 | 11 | 44 |

| 55 | 66 | 22 |

| 33 |

| 11 | 44 |

| 55 |

| 66 | 22 |

| 33 | 33 |
| min | max |

| 11 | 44 |

| 55 | 55 |

| 22 | 66 |

| 11 | 44 |
| min | max |

| 22 | 66 |

| 11 | 66 |
| min | max |

Logic used:

- The given list has more than two elements, so the algorithm divides the array from the middle and creates two subproblems.

- Both subproblems are treated as an independent problem and the same recursive process is applied to them.

- This division continues until subproblem size becomes one or two.

- If a_1 is the only element in the array, a_1 is the maximum and minimum.

- If the array contains only two elements a_1 and a_2, then the single comparison between two elements can decide the minimum and maximum of them.

Time Complexity:

The recurrence is for min-Max algorithm is:

$T(n) = 0,$ if n = 1

$T(n) = 1,$ if n = 2

$T(n) = 2T(n/2) + 2,$ if n > 2

$T(n) = 2T(n/2) + 2 \ldots (1)$

Substituting n by (n / 2) in Equation (1)

$T(n/2) = 2T(n/4) + 2$

$\quad = T(n) = 2(2T(n/4) + 2) + 2$

$\quad = 4T(n/4) + 4 + 2 \ldots (2)$

By substituting n by n/4 in Equation (1),

$\quad T(n/4) = 2T(n/8) + 2$

Substitute it in Equation (1),

$T(n) = 4[2T(n/8) + 2] + 4 + 2$

$\quad = 8T(n/8) + 8 + 4 + 2$

$\quad = 2^3 T(n/2^3) + 2^3 + 2^2 + 2^1$

$\quad \ldots\ldots$

$T(n) = 2^k T(n/2^k) + 2^k + \ldots + 2^2 + 2^1$

Assume $n/2^k = 2$ so $n/2 = 2^k$

$T(n) = 2^k T(2) + (2^k + \ldots + 2^2 + 2^1)$

$T(n) = 2^k T(2) + (2^1 + 2^2 + \ldots + 2^k)$

Using GP formula : $GP = a(r^k - 1)/(r-1)$

Here a = 2 and r = 2

$= 2^k + 2(2^k-1)/(2-1)$

$= 2^k + 2^{k+1} - 2$ { Assume $n/2^k = 2$ so $n/2 = 2^k$ and $n/2^{k+1} = 2(n/2) = n$ }

$= n/2 + n - 2$

$= 1.5 n - 2$

Time Complexity = O(n)

Algorithm:

DC_MAXMIN (A, low, high)

// Input: Array A of length n, and indices low = 0 and high = n-1

// Output: (min, max) variables holds minimum and maximum

if low = = high, Then   // low = = high

   return (A[low], A[low])

else if low = = high - 1 then   //low = = high - 1

     if A[low] < A[high] then

       return (A[low], A[high])

    else

       return (A[high], A[low])

   else

    mid ← (low + high) / 2

    [LMin, LMax] = DC_MAXMIN (A, low, mid)

    [RMin, RMax] = DC_MAXMIN (A, mid + 1, high)

    // Combine solution

    if LMax > RMax, Then

      max ← LMax

    else

      max ← RMax

  end

   if LMin < RMin, Then   // Combine solution.

      min ← LMin

    else

      min ← RMin

    end

  return (min, max)

end

Code:

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int n, i;
    int min, max;
    clrscr();
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements:\n");
    for(i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    min = arr[0];
    max = arr[0];

    for(i = 1; i < n; i++)
    {
        if(arr[i] > max)
        {
            max = arr[i];
        }
        else if(arr[i] < min)
        {
            min = arr[i];
        }
    }

    printf("The maximum element : %d\n", max);
    printf("The minimum element : %d\n", min);
    getch();
    return 0;
}
```

```c
#include<stdio.h>
#include<conio.h>

int main()
{
    int n, i;
    int min, max;
    clrscr();
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[100];
    printf("Enter the elements:\n");
    for(i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }

    min = arr[0];
    max = arr[0];

    for(i = 1; i < n; i++)
```

```c
    min = arr[0];
    max = arr[0];

    for(i = 1; i < n; i++)
    {
        if(arr[i] > max)
        {
            max = arr[i];
        }
        else if(arr[i] < min)
        {
            min = arr[i];
        }
    }

    printf("The maximum element : %d\n", max);
    printf("The minimum element : %d\n", min);
    getch();
    return 0;
}
```

Output:

Enter the number of elements : 7
Enter the elements :
9 13 4 30 8 1 18
The maximum element : 30
The minimum element : 1

```
Enter the number of elements: 7
Enter the elements:
9 13 4 30 8 1 18
The maximum element : 30
The minimum element : 1
```

Conclusion:

The minmax algorithm is a powerful technique for finding maximum and

minimum values in an array. The divide and conquer approach, which divides

the array into two halves and recursively finds the maximum and minimum

numbers, is more efficient. It has a time complexity of O(n) compared to the

naïve method, which requires comparing elements.

This approach can be further optimized by using a different data structure like a

binary search tree or heap. Overall, the minmax algorithm is a valuable tool for

large datasets.