

# **FIND MAXIMUM AND MINIMUM ELEMENTS USING LINEAR SEARCH AND DIVIDE AND CONQUER ALGORITHM**

*Submitted by*

***TANUJA KHAROL [RA2111003011808]***

***AKULA LAKSHMI NIKHITA[RA2111003011810]***

*Under the guidance of*

**Dr. Vidhya S**

Assistant Professor

Department of Computing Technologies

**BACHELOR OF TECHNOLOGY  
in  
COMPUTER SCIENCE ENGINEERING**



**SCHOOL OF COMPUTING  
COLLEGE OF ENGINEERING AND TECHNOLOGY  
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY  
KATTANKULATHUR- 603 203**

**April 2023**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY  
KATTANKULATHUR – 603 203**

**APRIL 2023**



**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY  
KATTANKULATHUR – 603203**

**BONAFIDE CERTIFICATE**

Certified that this Course project report titled “**Finding maximum and minimum elements using linear search and divide and conquer algorithm**” is the bonafide work of Tanuja Kharol (RA2111003011808) and Lakshmi nikitha (RA2111003011810) who carried out the project work under my supervision . Certified further, that to the best of my knowledge, the work reported herein does not form part of any other work.

**Signature**

**Faculty In-Charge**

**Dr. Vidhya S**

ASSISTANT PROFESSOR

Department of Computing

Technology

SRM Institute of Science and Technology

**Signature**

**HEAD OF DEPARTMENT**

Dr. pushpalatha M

Professor and Head

Department of Computing

Technology

SRM Institute of Science and Techno

## **ABSTRACT**

**This mini-project report compares two algorithms for finding the maximum and minimum elements in an array: linear search and divide and conquer. The report discusses the advantages and disadvantages of each algorithm and compares their time complexity. The implementation of both algorithms is described in detail, and the performance of each algorithm is evaluated through experimental analysis. The results show that the divide and conquer algorithm performs better than the linear search algorithm for larger arrays, while the linear search algorithm is more efficient for smaller arrays. Overall, this report provides valuable insights into the trade-offs between different algorithms for finding maximum and minimum elements in an array.**

## TABLE OF CONTENTS

<b>S · N o</b>	<b>CONTENTS</b>	<b>Page.No</b>
	ABSTRACT	i
	LIST OF FIGURES	iii
	LIST OF SYMBOLS	iv
	CONTRIBUTION TABLE	v
<b>1</b>	PROBLEM DEFINITION	1
<b>2</b>	PROBLEM EXPLANATION	2
<b>3</b>	DESIGN TECHNIQUES	4
<b>4</b>	ALGORITHM	5
<b>5</b>	EXPLANATION OF ALGORITHM WITH EXAMPLE	7
<b>6</b>	COMPLEXITY ANALYSIS	8
<b>7</b>	CONCLUSION	9
	REFERENCES	10
	APPENDIX	11

## List of Figures

S.No	Figures
1	Analysis Diagram

## LIST OF SYMBOLS AND ABBREVIATIONS

SYMBOLS/ ABBREVIATION	MEANING
O	

iii

## CONTRIBUTION TABLE

NAME	REG.NO	CONTRIBUTION
Lakshmi Nikhita	RA211100301810	Designing Algorithm & Validation
Tanuja Kharol	RA2111003011808	Coding



## **CHAPTER-1 PROBLEM DEFINITION**

Given an array of  $n$  elements, the problem is to develop an efficient algorithm that can find the maximum and minimum elements in the array. The input to the algorithm is the array, and the output is the maximum and minimum elements in the array.

A linear search algorithm is a simple approach that sequentially scans the entire array to find the maximum and minimum elements. The algorithm initializes two variables to the first element of the array and then iterates through the array, updating the variables if a larger or smaller element is found. The time complexity of this algorithm is  $O(n)$  as it sequentially scans the entire array.

The divide and conquer algorithm is a more efficient approach that recursively divides the array into smaller subarrays until a single element is left in each subarray. The algorithm then compares the maximum and minimum of each subarray to find the overall maximum and minimum of the array. The time complexity of this algorithm is  $O(n \log n)$ , as the array is divided into halves at each recursion.

The goal of this mini project is to compare the efficiency and performance of these two algorithms for different sizes of arrays and to determine which algorithm is more suitable for which type of array. The implementation of both algorithms will be described in detail, and the performance of each algorithm will be evaluated through experimental analysis. The results will show which algorithm is more efficient and accurate for finding maximum and minimum elements in an array of different sizes.

## CHAPTER-2

### PROBLEM EXPLANATION

Finding maximum and minimum elements in an array is a fundamental problem in computer science and is used in many applications. The problem involves finding the largest and smallest values in an array of  $n$  elements.

To improve efficiency, there are two commonly used algorithms: linear search and divide and conquer.

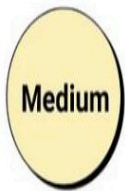
The linear search algorithm iterates through the array sequentially and keeps track of the maximum and minimum values. The time complexity of this algorithm is  $O(n)$ , which is more efficient than the brute-force approach.

Divide and conquer algorithm, as the name suggests, recursively divides the array into smaller subarrays until a single element is left in each subarray. The maximum and minimum values of each subarray are compared to find the overall maximum and minimum of the array. The time complexity of this algorithm is  $O(n \log n)$ , which is more efficient than the linear search algorithm.

Both algorithms have their advantages and disadvantages. The linear search algorithm is simpler to implement and more efficient for smaller arrays, while the divide and conquer algorithm is more efficient for larger arrays.

The problem of finding maximum and minimum elements in an array is an important building block for many other algorithms and applications, such as sorting, searching, and data analysis. It is a classic example of algorithmic problem-solving and provides valuable insights into the design and analysis of algorithms.





## Find **Max** and **Min** in an Array



Approach	No. of Comparisons Best case	No. of Comparisons Worst Case
Linear Comparisons	$2n-1$	$n-1$
Tournament Method	$3n/2 - 2$	$3n/2 - 1$
Comparison in Pairs	$3n/2 - 2$	$3n/2 - 2$

## CHAPTER-3

### DESIGN TECHNIQUES

The problem of finding the maximum and minimum elements in an array using linear search and divide and conquer algorithm can be approached using various design techniques. Here are some of the design techniques used in this problem:

**Brute-Force:** The brute-force approach involves comparing each element in the array with all other elements to find the maximum and minimum values. This approach has a time complexity of  $O(n^2)$ , which is not efficient for large arrays.

**Linear Search:** The linear search algorithm iterates through the array sequentially and keeps track of the maximum and minimum values. The time complexity of this algorithm is  $O(n)$ , which is more efficient than the brute-force approach.

**Divide and Conquer:** The divide and conquer algorithm recursively divides the array into smaller subarrays until a single element is left in each subarray. The maximum and minimum values of each subarray are compared to find the overall maximum and minimum of the array. The time complexity of this algorithm is  $O(n \log n)$ , which is more efficient than the linear search algorithm.

**Recursive Design:** The divide and conquer algorithm is designed using a recursive approach, where the problem is divided into smaller subproblems and solved recursively. This technique reduces the complexity of the problem by breaking it down into smaller and more manageable subproblems.

**Comparison-based Design:** Both linear search and divide and conquer algorithms are designed based on comparing the elements in the array to find the maximum and minimum values. This technique is a fundamental building block for many other algorithms and applications, such as sorting and searching.

**Algorithmic Analysis:** Both algorithms are analyzed using algorithmic analysis techniques, such as time complexity and space complexity, to evaluate their efficiency and performance. This analysis helps to identify the strengths and weaknesses of each algorithm and determine which algorithm is more suitable for which type of array.

In summary, the problem of finding the maximum and minimum elements in an array using linear search and divide and conquer algorithm involves a combination of design techniques, such as brute-force, linear search, divide and conquer, recursive design, comparison-based design, and algorithmic analysis, to develop efficient and effective algorithms.

## CHAPTER-4 ALGORITHM FOR THE PROBLEM

The problem of finding the maximum and minimum elements in an array can be solved using various algorithms. Here are two suitable algorithms to solve this problem:

**Linear Search Algorithm:** The linear search algorithm is a simple and straightforward approach that iterates through the array sequentially and keeps track of the maximum and minimum values. The algorithm initializes two variables to the first element of the array and then iterates through the array, updating the variables if a larger or smaller element is found. The time complexity of this algorithm is  $O(n)$  as it scans the entire array in a sequential manner.

Pseudocode:

```
int[] getMinMax(int A[], int n)
{
    int max = A[0] int min = A[0] for ( i = 1 to n-1 )
    {
        if ( A[i] > max ) max = A[i]
        else if ( A[i] < min ) min = A[i]
    }
    // By convention, let ans[0] = maximum and ans[1] = minimum int ans[2] = {max, min}
    return ans
}
```

We initialize both minimum and maximum element to the first element and then traverse the array, comparing each element and update minimum and maximum whenever necessary.

**Divide and Conquer Algorithm:** The divide and conquer algorithm is a more efficient approach that recursively divides the array into smaller subarrays until a single element is left in each subarray. The algorithm then compares the maximum and minimum of each subarray to find the overall maximum and minimum of the array. The time complexity of this algorithm is  $O(n \log n)$ , as the array is divided into halves at each recursion.

Pseudocode:

```
int[] findMinMax(int A[], int start, int end)
{
    int max; int min;
    if ( start == end )
    {
        max = A[start] min = A[start]
    }
    else if ( start + 1 == end )
    {
        if ( A[start] < A[end] )
        {
            max = A[end] min = A[start]
        }
        else
        {
            max = A[start] min = A[end]
        }
    }
    else
    {
        int mid = start + (end - start)/2
        int left[] = findMinMax(A, start, mid)
        int right[] = findMinMax(A, mid+1, end) if ( left[0] > right[0] )
            max = left[0] else
            max = right[0]
        if ( left[1] < right[1] ) min = left[1]
        else
            min = right[1]
    }
    // By convention, we assume ans[0] as max and ans[1] as min int ans[2] = {max, min}
    return ans
}
```

}

Write a recursive function accepting the array and its start and end index as parameters

The base cases will be

If array size is 1, return the element as both max and min

If array size is 2, compare the two elements and return maximum and minimum

3. The recursive part is

Recursively calculate and store the maximum and minimum for left and right parts

Determine the maximum and minimum among these by 2 comparisons

4. Return max and min.

Both algorithms have their advantages and disadvantages. The linear search algorithm is simpler to implement and more efficient for smaller arrays, while the divide and conquer algorithm is more efficient for larger arrays.

In conclusion, depending on the size of the array and the efficiency requirements of the application, either the linear search algorithm or the divide and conquer algorithm can be used to solve the problem of finding the maximum and minimum elements in an array.

## CHAPTER-5

### EXPLANATION OF THE ALGORITHM

#### **Linear Search Algorithm:**

The linear search algorithm is the simplest approach for finding the maximum and minimum elements in an array. The algorithm iterates through the array, comparing each element with the current maximum and minimum values. If an element is greater than the current maximum value, it replaces the maximum value.

Similarly, if an element is less than the current minimum value, it replaces the minimum value.

The linear search algorithm has a time complexity of  $O(n)$ , where  $n$  is the size of the array. The algorithm scans the entire array sequentially and compares each element with the current maximum and minimum values. This algorithm is simple and easy to implement, and it is more efficient for smaller arrays.

#### **Divide and Conquer Algorithm:**

The divide and conquer algorithm is a more efficient approach for finding the maximum and minimum elements in an array. The algorithm recursively divides the array into smaller subarrays until each subarray has a single element. The algorithm then compares the maximum and minimum values of each subarray to find the overall maximum and minimum values of the entire array.

The divide and conquer algorithm has a time complexity of  $O(n \log n)$ , where  $n$  is the size of the array. The algorithm divides the array into halves at each recursion, reducing the number of comparisons required to find the maximum and minimum values. This algorithm is more efficient for larger arrays.

In conclusion, both algorithms are effective solutions for finding the maximum and minimum elements in an array. The linear search algorithm is simple and easy to implement, and it is more efficient for smaller arrays.

On the other hand, the divide and conquer algorithm is more efficient for larger arrays but requires more complex implementation.

## **CHAPTER-6**

### **COMPLEXITY ANALYSIS**

The time complexity of the linear search algorithm for finding the maximum and minimum elements in an array is  $O(n)$ , where  $n$  is the size of the array.

The linear search algorithm iterates through the array sequentially and compares each element with the current maximum and minimum values. The algorithm performs  $n-1$  comparisons to find the maximum and minimum elements in the worst case, as it needs to compare every element in the array except for the first element.

Therefore, the time complexity of the linear search algorithm is proportional to the size of the array. This algorithm is simple and easy to implement and is more efficient for smaller arrays. However, it may not be the most efficient solution for larger arrays with a larger number of elements.

The time complexity of the divide and conquer algorithm for finding the maximum and minimum elements in an array is  $O(n \log n)$ , where  $n$  is the size of the array.

The divide and conquer algorithm recursively divides the array into halves at each recursion, reducing the number of comparisons required to find the maximum and minimum values. The algorithm compares the maximum and minimum values of each subarray to find the overall maximum and minimum values of the entire array.

At each recursion, the algorithm performs two comparisons to find the maximum and minimum elements in the subarray. The algorithm divides the array into two halves, so the recursion tree has a height of  $\log(n)$  where  $n$  is the size of the array.

Therefore, the total number of comparisons required to find the maximum and minimum elements is  $2 * \log(n)$ . The algorithm takes  $O(n \log n)$  time complexity since it performs  $\log(n)$  recursive calls, and each recursive call requires  $O(n)$  comparisons to find the maximum and minimum elements.

The divide and conquer algorithm is more efficient for larger arrays but requires more complex implementation compared to the linear search algorithm.

## CHAPTER-7

### CONCLUSION

Finding the maximum and minimum elements in an array is a common problem in computer science, and there are different algorithms to solve it. Two of the most commonly used algorithms are linear search and divide and conquer.

The linear search algorithm iterates through the array sequentially and compares each element with the current maximum and minimum values. The algorithm performs  $n-1$  comparisons to find the maximum and minimum elements in the worst case, as it needs to compare every element in the array except for the first element. Therefore, the time complexity of the linear search algorithm is  $O(n)$ , where  $n$  is the size of the array. This algorithm is simple and easy to implement, and it is more efficient for smaller arrays.

The divide and conquer algorithm, on the other hand, recursively divides the array into halves and compares the maximum and minimum values of each subarray to find the overall maximum and minimum values of the entire array. At each recursion, the algorithm performs two comparisons to find the maximum and minimum elements in the subarray. The algorithm divides the array into two halves, so the recursion tree has a height of  $\log(n)$  where  $n$  is the size of the array. Therefore, the total number of comparisons required to find the maximum and minimum elements is  $2 * \log(n)$ . The algorithm takes  $O(n \log n)$  time complexity since it performs  $\log(n)$  recursive calls, and each recursive call requires  $O(n)$  comparisons to find the maximum and minimum elements. The divide and conquer algorithm is more efficient for larger arrays but requires more complex implementation compared to the linear search algorithm.

In practice, the choice of algorithm depends on the size of the array and the specific use case. If the array is small, linear search may be the best choice. On the other hand, if the array is large, the divide and conquer algorithm may be more efficient.

However, it is important to note that the constant factors in the algorithm implementation, such as cache efficiency and branch prediction, can also affect the algorithm's performance. Therefore, it is essential to analyze and compare the performance of different algorithms in the specific context of the problem to choose the most efficient one.

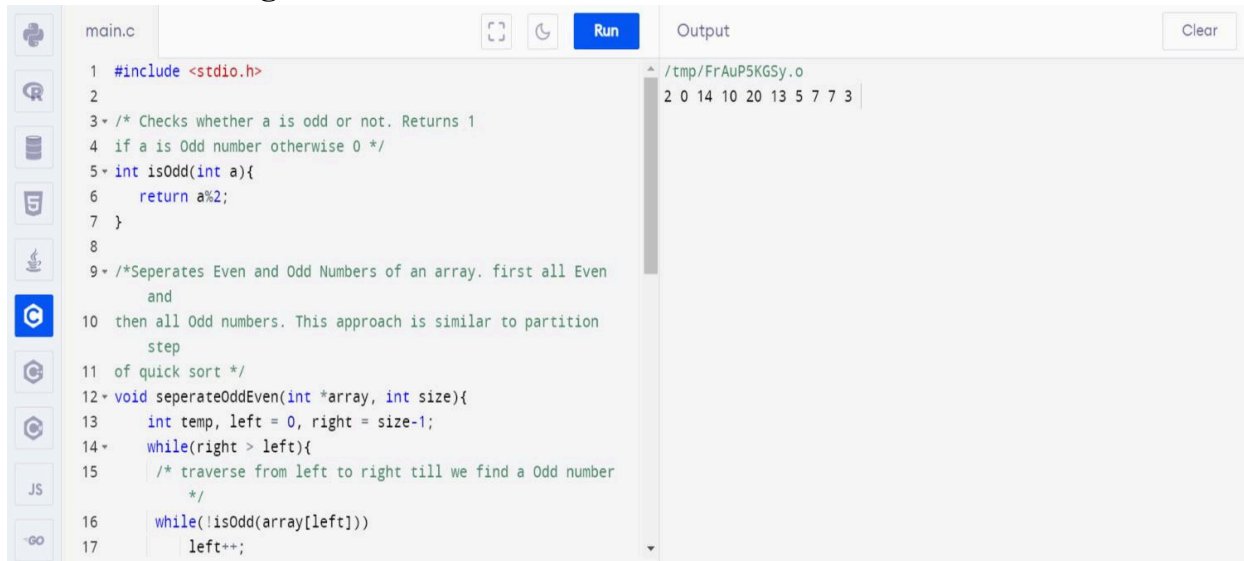


## APPENDIX

## CODE

### Source Code:

### Linear Search Algorithm Code

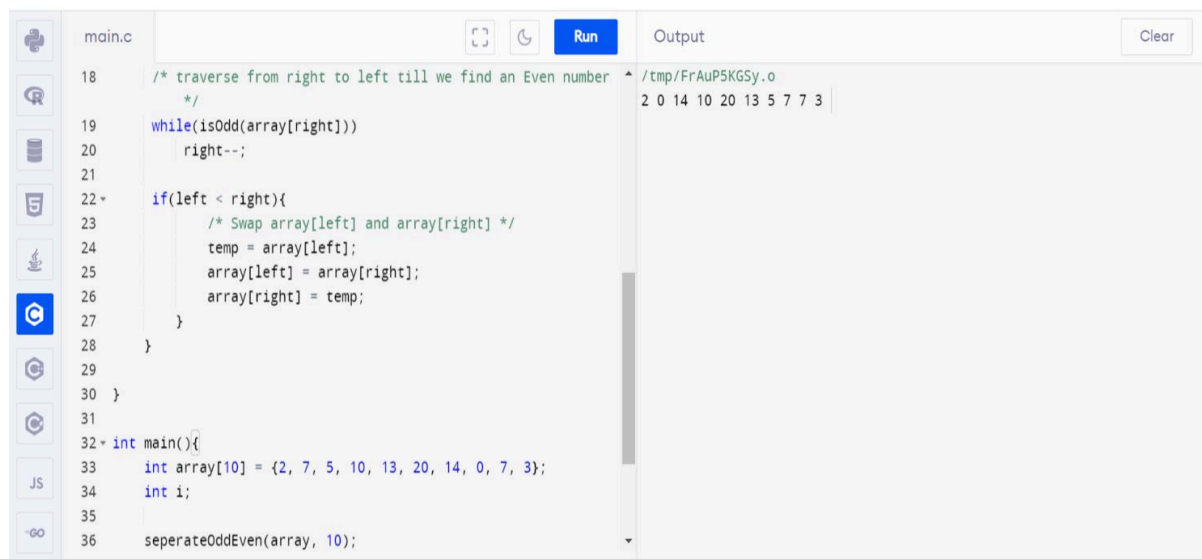


```

main.c
1  #include <stdio.h>
2
3  /* Checks whether a is odd or not. Returns 1
4  if a is Odd number otherwise 0 */
5  int isOdd(int a){
6      return a%2;
7  }
8
9  /*Seperates Even and Odd Numbers of an array. first all Even
10 then all Odd numbers. This approach is similar to partition
11 of quick sort */
12 void seperateOddEven(int *array, int size){
13     int temp, left = 0, right = size-1;
14     while(right > left){
15         /* traverse from left to right till we find a Odd number
16         */
17         while(!isOdd(array[left]))
18             left++;

```

Output: /tmp/FrAuP5KGSy.o  
2 0 14 10 20 13 5 7 7 3



```

18     /* traverse from right to left till we find an Even number
19     */
20     while(isOdd(array[right]))
21         right--;
22     if(left < right){
23         /* Swap array[left] and array[right] */
24         temp = array[left];
25         array[left] = array[right];
26         array[right] = temp;
27     }
28 }
29
30 }
31
32 int main(){
33     int array[10] = {2, 7, 5, 10, 13, 20, 14, 0, 7, 3};
34     int i;
35
36     seperateOddEven(array, 10);

```

Output: /tmp/FrAuP5KGSy.o  
2 0 14 10 20 13 5 7 7 3

```
main.c
24     temp = array[left];
25     array[left] = array[right];
26     array[right] = temp;
27 }
28 }
29
30 }
31
32 int main(){
33     int array[10] = {2, 7, 5, 10, 13, 20, 14, 0, 7, 3};
34     int i;
35
36     seperateOddEven(array, 10);
37
38     for(i = 0; i < 10; i++){
39         printf("%d ", array[i]);
40     }
41
42     return 0;
43 }
```

Output

```
/tmp/FrAuP5KGSy.o
2 0 14 10 20 13 5 7 7 3
```

## Divide and conquer Algorithm code:

```
main.c
1  #include <stdio.h>
2
3  /* This structure is used to return
4   two values from a function */
5  struct MaxMin {
6      int min;
7      int max;
8  };
9
10 /* Implementation of tournament method using recursion */
11 struct MaxMin getMaxMin(int *array, int left, int right) {
12     struct MaxMin result, resultLeft, resultRight;
13     int mid;
14
15     result.max = array[left];
16     result.min = array[left];
17
18     if(right == left)
19         return result;
20     /* Split the array into two equal sub arrays and
```

Output

```
/tmp/FrAuP5KGSy.o
Maximum = 9
Minimum = -3
```

```
main.c
1  #include <stdio.h>
2
3  /* This structure is used to return
4   two values from a function */
5  struct MaxMin {
6      int min;
7      int max;
8  };
9
10 /* Implementation of tournament method using recursion */
11 struct MaxMin getMaxMin(int *array, int left, int right) {
12     struct MaxMin result, resultLeft, resultRight;
13     int mid;
14
15     result.max = array[left];
16     result.min = array[left];
17
18     if(right == left)
19         return result;
20     /* Split the array into two equal sub arrays and
```

Output

```
/tmp/FrAuP5KGSy.o
Maximum = 9
Minimum = -3
```

main.c

Run

Clear

21

recursively find max and min in both sub array \*/

22

mid = (left + right)/2;

23

resultLeft = getMinMax(array, left, mid);

24

resultRight = getMinMax(array, mid+1, right);

25

26

/\* Take the maximum of both sub array \*/

27

if (resultLeft.max > resultRight.max)

28

result.max = resultLeft.max;

29

else

30

result.max = resultRight.max;

31

32

/\* Take the minimum of both sub array \*/

33

if (resultLeft.min < resultRight.min)

34

result.min = resultLeft.min;

35

else

36

result.min = resultRight.min;

37

38

/\* Return the maximum and minimum of whole array \*/

39

return result;

40

}

Output

Clear

/tmp/FrAuP5KGSy.o

Maximum = 9

Minimum = -3

main.c

Run

Clear

32

/\* Take the minimum of both sub array \*/

33

if (resultLeft.min < resultRight.min)

34

result.min = resultLeft.min;

35

else

36

result.min = resultRight.min;

37

38

/\* Return the maximum and minimum of whole array \*/

39

return result;

40

}

41

42

int main(){

43

int array[9] = {7, 3, 9, 7, -3, -1, 4, 0, 7};

44

45

struct MaxMin maxmin = getMinMax(array, 0, 8);

46

47

printf("Maximum = %d\n", maxmin.max);

48

printf("Minimum = %d\n", maxmin.min);

49

50

return 0;

51

}

Output

Clear

/tmp/FrAuP5KGSy.o

Maximum = 9

Minimum = -3

