BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE ENGINEERING

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR - 603203
JUNE 2022

PROJECT REPORT

Submitted by:	Milind Srivastava	(RA2011003011019)
	Aryan Chakraborty	(RA2011003011043)
	Shikhar Pandey	(RA2011003011018)

SRM

INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University u/s 3 of UGC Act, 1956)

School of Computing

SRM IST, Kattankulathur – 603 203

Course Name: Design Analysis of Algorithm

Course Code: 18CSC204J

Project Title	**Food Challenge**
Team Members	Shikhar Pandey (RA2011003011018) Milind Srivastava (RA2011003011019) Aryan Chakraborty (RA2011003011043)

Course Faculty Signature

Index table

Contribution Table

S.No	Topics	Team Member
1.	Problem Definition	Milind Shikhar
2.	Problem Structure	Milind Shikhar
3.	Designed Technique Used	Team
4.	Algorithm for the problem	Aryan Milind
5.	Explanation of algorithm	Aryan Shikhar
6.	Code Snippet	Team
7.	Sample Output	Aryan Shikhar
8.	Complexity Analysis	Aryan Milind
9.	Conclusion	Team
10.	Report	Team

1. Problem Statement:

FOOD CHALLENGE

Using - Divide and Conquer

A team consisting of A people who take 1 unit of time to completely finish a bowl of soup. There is an array of integers C of size N representing the size and number of bowls respectively. We need to find the minimum time in which the team can finish eating all the bowls in order to win the eating challenge.

There are some constraints that need to be kept in mind while devising the solution to this particular problem are :-

a. 2 eater/ people cannot share a bowl to eat.

b. An eater will only eat a contiguous bowl.

2. Problem Structure:-

- Given an integer A and an array of integers C of size N.
- Element C[i] represents the length/size of the i th bowl.
- You have to eat all N bowls [C0, C1, C2, C3 … CN-1].
- There are A eaters available.
- 2 Eaters cannot share a bowl to eat.
- An eater will only eat a contiguous bowl.
- Each of them takes 1 unit of time to eat 1 unit of bowl.
- The time required for A eater to finish the soup of bowl is equal to the summation of the size of the bowl
- Calculate and return minimum time required to eat all bowl under the constraints that any eater will only eat contiguous sections of eater.

3. Designed Technique Used

Divide and Conquer

Is an algorithmic pattern. In algorithmic methods, the design is to take a dispute on a huge input, break the input into minor pieces, decide the problem on each of the small pieces, and then merge the piecewise solutions into a global solution.

This mechanism of solving the problem is called the Divide & Conquer Strategy.

Advantages:

Solving : Divide and conquer is a powerful tool for solving conceptually difficult problems: all it requires is a way of breaking the problem into sub-problems, of solving the trivial cases, and of combining sub-problems to the original problem.

Algorithm Efficiency:

The divide-and-conquer paradigm often helps in the discovery of efficient algorithms.The D&C approach led to an improvement in the asymptotic cost of the solution.

Parallelism:

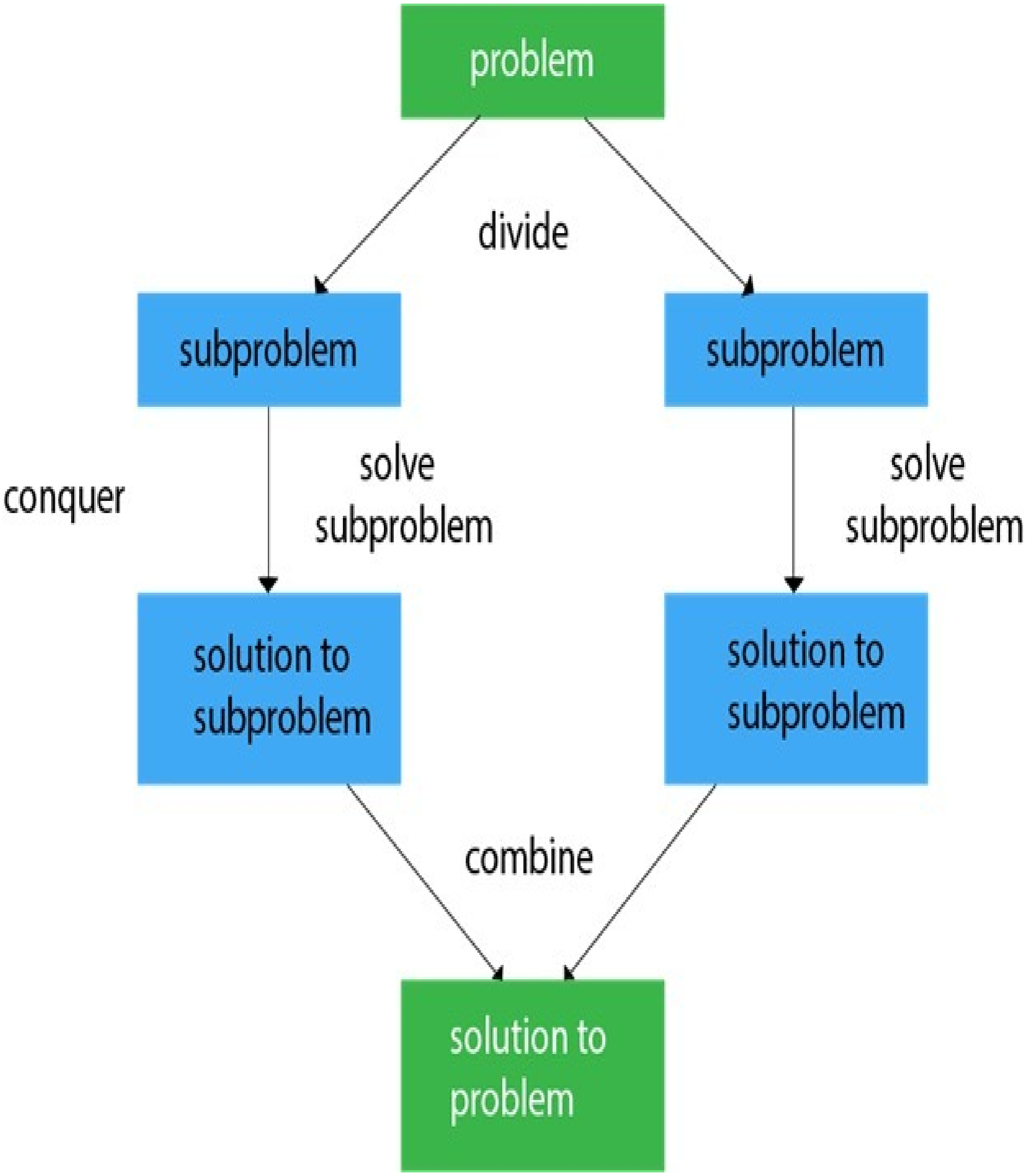
Divide-and-conquer algorithms are naturally adapted for execution in multi-processor machines, especially shared-memory systems where the communication of data between processors does not need to be planned in advance because distinct sub-problems can be executed on different processors.

Memory Access:

Divide-and-conquer algorithms naturally tend to make efficient use of memory caches. The reason is that once a sub-problem is small enough, it and all its sub-problems can, in principle, be solved within the cache, without accessing the slower main memory.

Divide and Conquer algorithm consists of a dispute using the following three steps.

- Divide the original problem into a set of subproblems.
- Conquer: Solve every subproblem individually, recursively.
- Combine: Put together the solutions of the subproblems to get the solution to the whole problem.



Graphical Representation of Divide and Conquer

4. Algorithm For the Problem

int getMax(int arr[], int n)

```
for (int i = 0; i < n; i++)

if (arr[i] > max)

max = arr[i];

return max;

// return the maximum element from the array
```

int getSum(int arr[], int n)

```
total = 0;

for ( i = 0; i < n; i++)

total += arr[i];

return total;

END

 // return the sum of the elements in the array
```

int numberOfEater(int arr[], int n, int maxLen)

```
total = 0, numEater = 1;
START for ( i = 0; i < n; i++)

    total += arr[i];
    IF total > maxLen
        total = arr[i];
        numEater++;
         // for next count
    END IF
END FOR
return numEater;
END
```

int partition(int arr[], int n, int k)

```
lo = getMax(arr, n);
hi = getSum(arr, n);
START While lo < hi :
    mid = lo + (hi - lo) / 2;
    int requiredEater = numberOfEater(arr, n, mid);
    IF requiredEater <= k)
```

```
hi = mid;
ELSE
lo = mid + 1;
END While
return lo   // required
END
END PROGRAM
```

5. Algorithm Explanation:-

The given question requires us obtain the minimum time that the team takes to finish eating

- We divide the array of bowls and obtain the max element in each subarray.
- The highest possible value in the range is the sum of all the elements in the array and this happens when we allot 1 person to eat all the bowls. The lowest possible value of this range is the maximum value, as in this allocation we can allot max to one eater and divide the other bowls such that the total food is less than or equal to max.

Eg: Input : k = 3, A = {1,2,3,4,5}

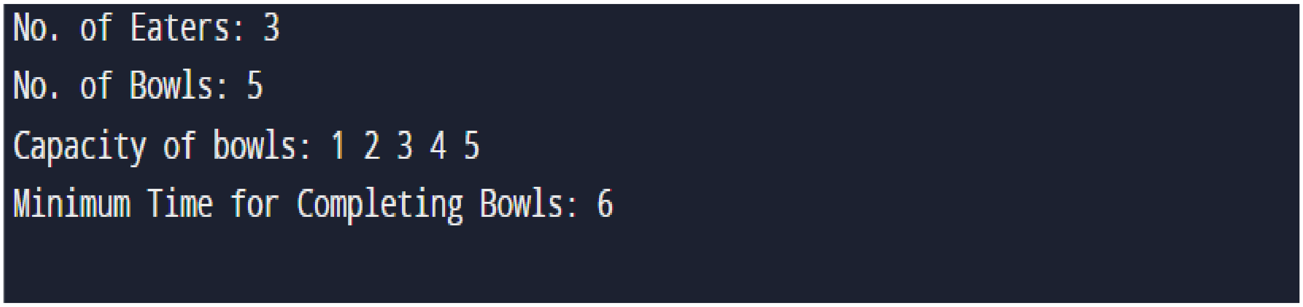
Output : 6

Code Snippet :

```
#include <iostream>

#include <climits>

using namespace std;

int getMax(int arr[], int n)

{

    int max = INT_MIN;

    for (int i = 0; i < n; i++)

        if (arr[i] > max)

            max = arr[i];

    return max;

}

int getSum(int arr[], int n)

{

    int total = 0;

    for (int i = 0; i < n; i++)

        total += arr[i];

    return total;

}

int numberOfEaters(int arr[], int n, int maxLen)

{
    int total = 0, numEaters = 1;
     for (int i = 0; i < n; i++) {
        total += arr[i];
         if (total > maxLen) {
             // for next count
            total = arr[i];
            numEaters++;
        }
    }
     return numEaters;
}
int partition(int arr[], int n, int k)
{    int lo = getMax(arr, n);
    int hi = getSum(arr, n);
    while (lo < hi) {
        int mid = lo + (hi - lo) / 2;
        int requiredEaters = numberOfEaters(arr, n, mid);
         if (requiredEaters <= k)
            hi = mid;
            else
            lo = mid + 1;  }
    // required
    return lo;
}
int main()
{   int k;
    cout<<"No. of Eaters: ";
    cin>>k;
    int n;
    cout<<"No. of Bowls: ";
    cin>>n;
    int arr[n];
    cout<<"Capacity of bowls: ";
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    cout<<"Minimum Time for Completing Bowls: ";
    cout << partition(arr, n, k) << endl;
    return 0;}
```

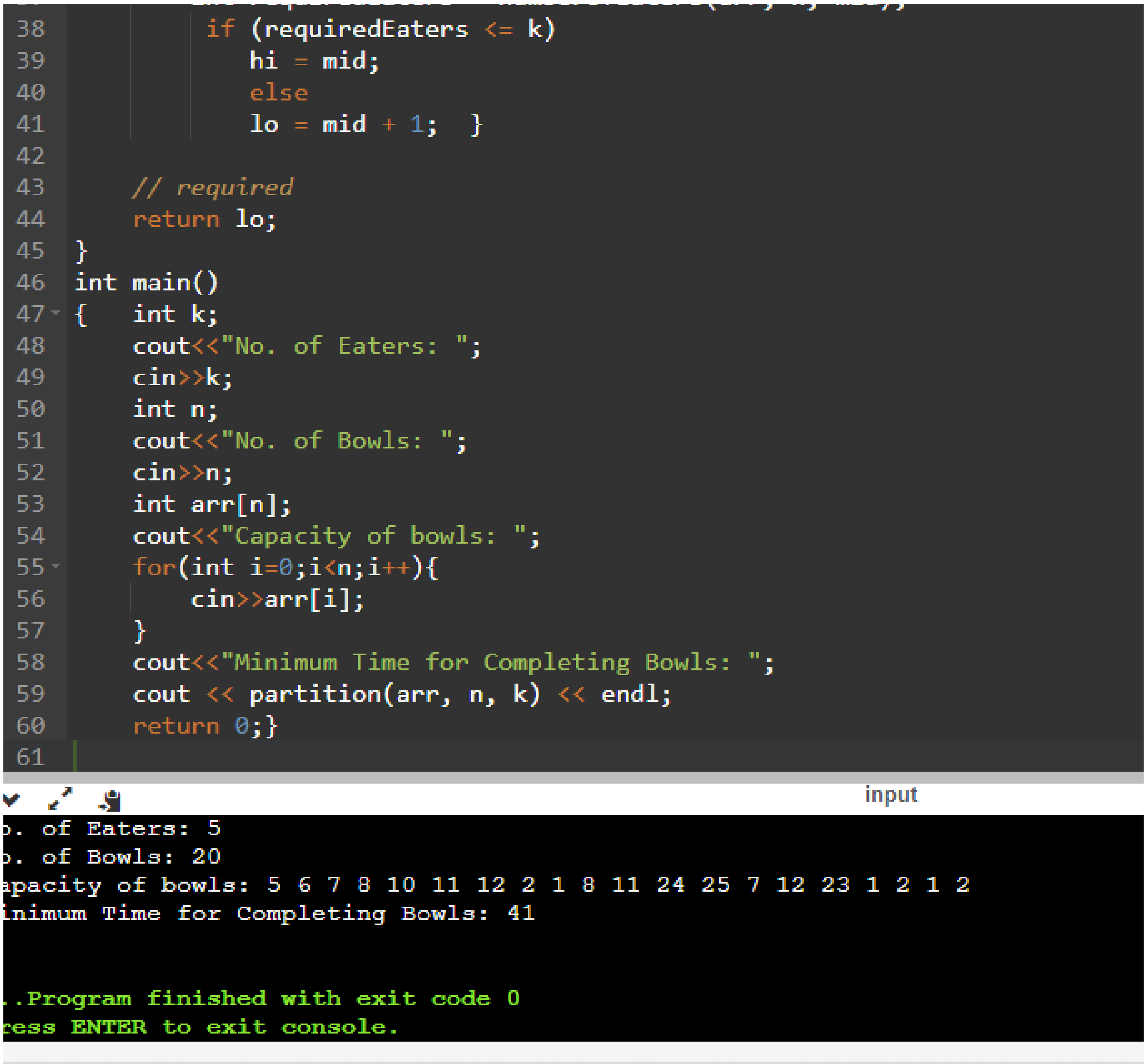
Test Cases :

Test Case 1:



```
No. of Eaters: 3
No. of Bowls: 5
Capacity of bowls: 1 2 3 4 5
Minimum Time for Completing Bowls: 6
```

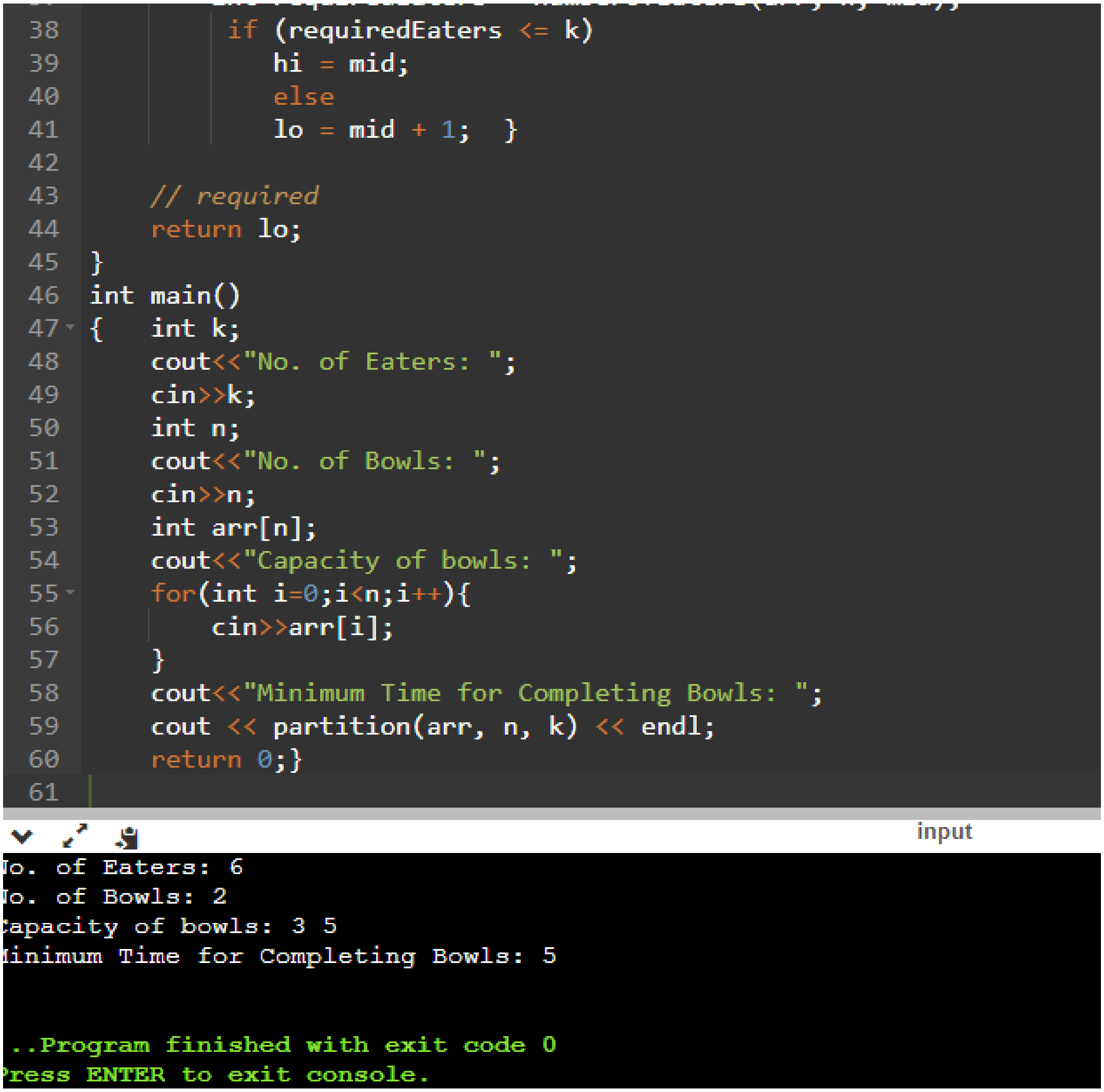
Test Case 2:



```
          if (requiredEaters <= k)
             hi = mid;
             else
             lo = mid + 1;   }

    // required
    return lo;
}
int main()
{   int k;
    cout<<"No. of Eaters: ";
    cin>>k;
    int n;
    cout<<"No. of Bowls: ";
    cin>>n;
    int arr[n];
    cout<<"Capacity of bowls: ";
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    cout<<"Minimum Time for Completing Bowls: ";
    cout << partition(arr, n, k) << endl;
    return 0;}
```

```
o. of Eaters: 5
o. of Bowls: 20
apacity of bowls: 5 6 7 8 10 11 12 2 1 8 11 24 25 7 12 23 1 2 1 2
inimum Time for Completing Bowls: 41


..Program finished with exit code 0
ress ENTER to exit console.
```

Test Case 3:



```
            if (requiredEaters <= k)
               hi = mid;
               else
               lo = mid + 1;   }

        // required
        return lo;
}
int main()
{    int k;
     cout<<"No. of Eaters: ";
     cin>>k;
     int n;
     cout<<"No. of Bowls: ";
     cin>>n;
     int arr[n];
     cout<<"Capacity of bowls: ";
     for(int i=0;i<n;i++){
         cin>>arr[i];
     }
     cout<<"Minimum Time for Completing Bowls: ";
     cout << partition(arr, n, k) << endl;
     return 0;}
```

```
o. of Eaters: 6
o. of Bowls: 2
apacity of bowls: 3 5
inimum Time for Completing Bowls: 5


..Program finished with exit code 0
ress ENTER to exit console.
```

6. Time Complexity

Time Complexity can be calculated using many ways method including

- Recursion tree
- Master's Theorem
- Substitution Method

Taking the help of these methods, tThe time complexity of the above approach is calculated to be **O(n*log(sum arr[])**

7. Conclusion

To solve the problem, the approach that was used was divide and conquer wherein the array is divided between maximum capacity bowl and the rest of the bowls remaining. Then further from this the subsequent array was divided again between the maximum capacity bowl remaining and again the remaining ones until we reach the point where both the arrays generated have only 1 element in each of them.

Using this approach we were able to find a solution to the **FOOD CHALLENEGE** problem and also executed several test cases to verify the same.

8. Reference

- Geeksforgeeks,org
- Leetcode.com
- Interviewbit.com