**Group A : DESIGN AND ANALYSIS ALGORITHM**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Assignment No: 1 Title Name:** Calculate Fibonacci numbers and find its step count. **Name**: Abhishek Santosh Bankar

**Class** : BE **Div**: 1 **Batch**: A

## Roll No: 405A011

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Program:









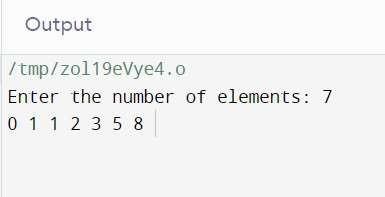


































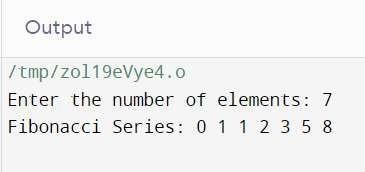














\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Assignment No: 2

**Title Name:** Write a program to implement Job sequencing with deadlines using a greedy method.

**Name**: Abhishek Santosh Bankar

**Class** : BE **Div**: 1 **Batch**: A

## Roll No: 405A011

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*









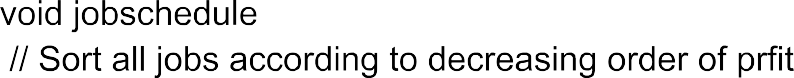


















































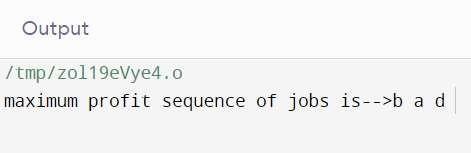












\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Assignment No: 3 Title Name:** Huffman Encoding using a greedy strategy **Name**: Abhishek Santosh Bankar

**Class** : BE **Div**: 1 **Batch**: A

## Roll No: 405A011

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Program:

// Huffman Coding in C++

#include <iostream> using namespace std; #define MAX\_TREE\_HT 50

struct MinHNode { unsigned freq; char item;

struct MinHNode \*left, \*right;

};

struct MinH { unsigned size; unsigned capacity;

struct MinHNode \*\*array;

};

// Creating Huffman tree node

struct MinHNode \*newNode(char item, unsigned freq)

{

struct MinHNode \*temp = (struct MinHNode \*)malloc(sizeof(struct MinHNode)); temp->left = temp->right = NULL;

temp->item = item; temp->freq = freq; return temp;

}

// Create min heap using given capacity struct MinH \*createMinH(unsigned capacity)

{

struct MinH \*minHeap = (struct MinH \*)malloc(sizeof(struct MinH)); minHeap->size = 0;

minHeap->capacity = capacity;

minHeap->array = (struct MinHNode \*\*)malloc(minHeap->capacity \* sizeof(struct MinHNode \*)); return minHeap;

}

// Print the array

void printArray(int arr[], int n)

{

int i;

for (i = 0; i < n; ++i) cout << arr[i];

cout << "\n";

}

// Swap function

void swapMinHNode(struct MinHNode \*\*a, struct MinHNode \*\*b)

{

struct MinHNode \*t = \*a;

\*a = \*b;

\*b = t;

}

// Heapify

void minHeapify(struct MinH \*minHeap, int idx)

{

int smallest = idx; int left = 2 \* idx + 1;

int right = 2 \* idx + 2;

if (left < minHeap->size && minHeap->array[left]->freq < minHeap->array[smallest]->freq) smallest = left;

if (right < minHeap->size && minHeap->array[right]->freq < minHeap->array[smallest]->freq) smallest = right;

if (smallest != idx) { swapMinHNode(&minHeap->array[smallest],

&minHeap->array[idx]); minHeapify(minHeap, smallest);

}

}

// Check if size if 1

int checkSizeOne(struct MinH \*minHeap)

{

return (minHeap->size == 1);

}

// Extract the min

struct MinHNode \*extractMin(struct MinH \*minHeap)

{

struct MinHNode \*temp = minHeap->array[0];

minHeap->array[0] = minHeap->array[minHeap->size - 1];

--minHeap->size; minHeapify(minHeap, 0); return temp;

}

// Insertion

void insertMinHeap(struct MinH \*minHeap, struct MinHNode \*minHeapNode)

{

++minHeap->size;

int i = minHeap->size - 1;

while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq) { minHeap->array[i] = minHeap->array[(i - 1) / 2];

i = (i - 1) / 2;

}

minHeap->array[i] = minHeapNode;

}

// BUild min heap

void buildMinHeap(struct MinH \*minHeap)

{

int n = minHeap->size - 1; int i;

for (i = (n - 1) / 2; i >= 0; --i) minHeapify(minHeap, i);

}

int isLeaf(struct MinHNode \*root) { return !(root->left) && !(root->right);

}

struct MinH \*createAndBuildMinHeap(char item[], int freq[], int size)

{

struct MinH \*minHeap = createMinH(size); for (int i = 0; i < size; ++i)

minHeap->array[i] = newNode(item[i], freq[i]);

minHeap->size = size; buildMinHeap(minHeap); return minHeap;

}

struct MinHNode \*buildHfTree(char item[], int freq[], int size)

{

struct MinHNode \*left, \*right, \*top;

struct MinH \*minHeap = createAndBuildMinHeap(item, freq, size); while (!checkSizeOne(minHeap)) {

left = extractMin(minHeap); right = extractMin(minHeap);

top = newNode('$', left->freq + right->freq); top->left = left;

top->right = right; insertMinHeap(minHeap, top);

}

return extractMin(minHeap);

}

void printHCodes(struct MinHNode \*root, int arr[], int top)

{

if (root->left) { arr[top] = 0;

printHCodes(root->left, arr, top + 1);

}

if (root->right) { arr[top] = 1;

printHCodes(root->right, arr, top + 1);

}

if (isLeaf(root)) {

cout << root->item << " | "; printArray(arr, top);

}

}

// Wrapper function

void HuffmanCodes(char item[], int freq[], int size)

{

struct MinHNode \*root = buildHfTree(item, freq, size); int arr[MAX\_TREE\_HT], top = 0;

printHCodes(root, arr, top);

}

int main()

{

char arr[] = {'A', 'B', 'C', 'D'};

int freq[] = {5, 1, 6, 3};

int size = sizeof(arr) / sizeof(arr[0]); cout << "Char | Huffman code "; cout << "\n \n"; HuffmanCodes(arr, freq, size);

}

## Output:

Char | Huffman code

C | 0

B | 100

D | 101

A | 11

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Assignment No: 4

**Title Name:** Solve a fractional Knapsack problem using a greedy method

**Name**: Abhishek Santosh Bankar

**Class** : BE **Div**: 1 **Batch**: A

## Roll No: 405A011

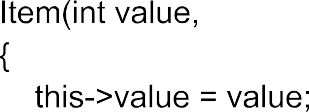
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Program:

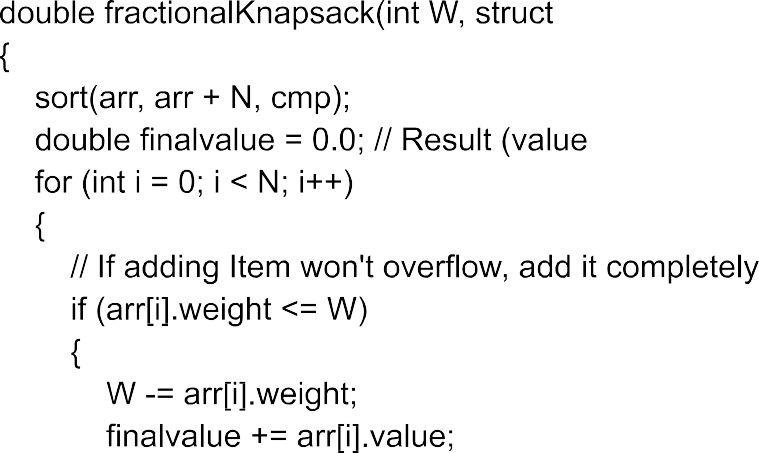
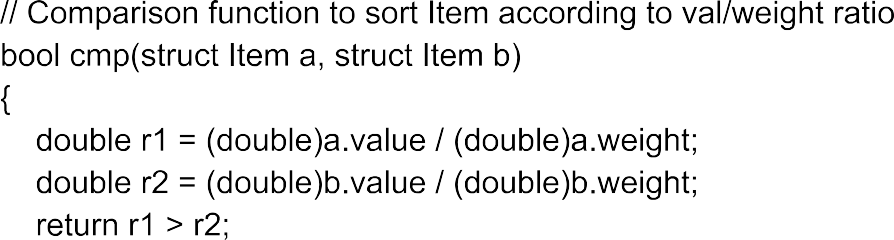










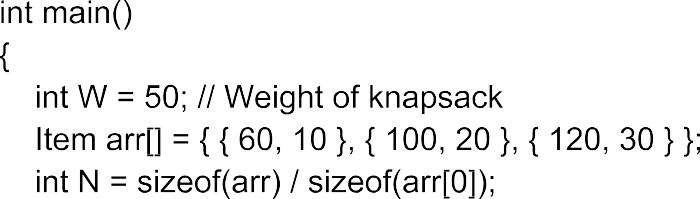




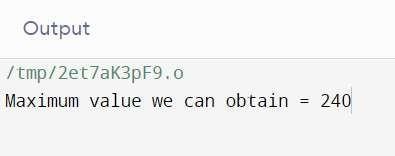




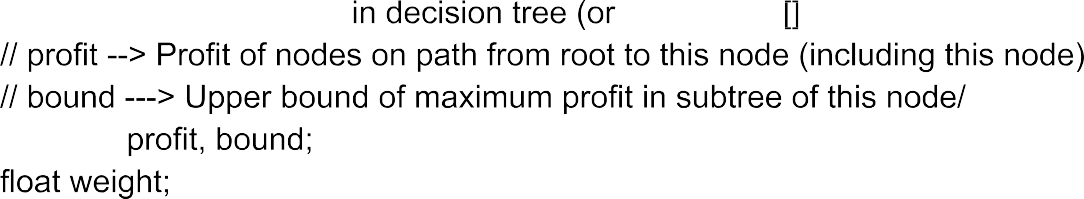








\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



## Assignment No: 5

**Title Name:** Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy

**Name**: Abhishek Santosh Bankar

**Class** : BE **Div**: 1 **Batch**: A

## Roll No: 405A011

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

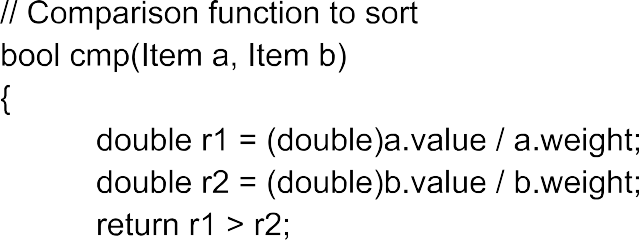
## Program



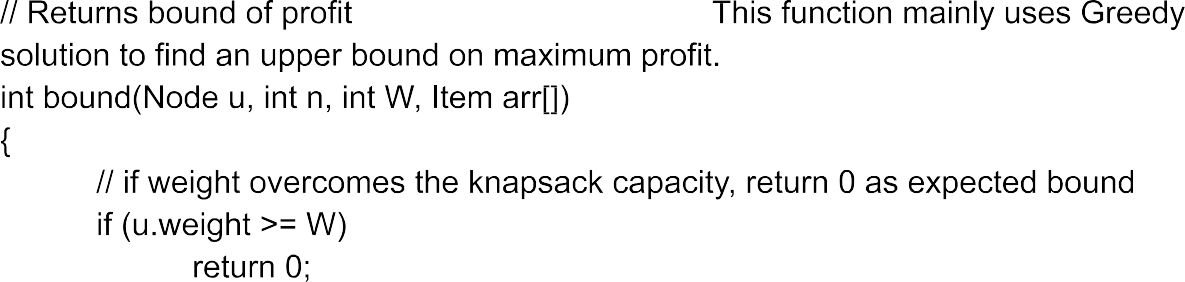








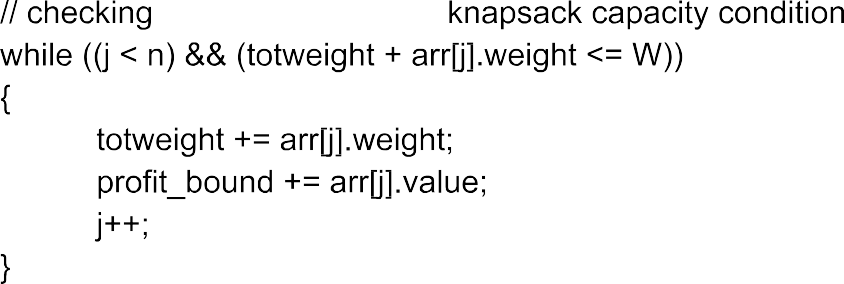


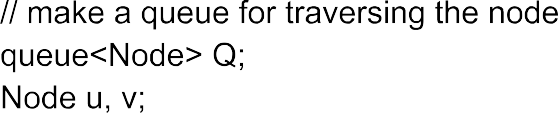
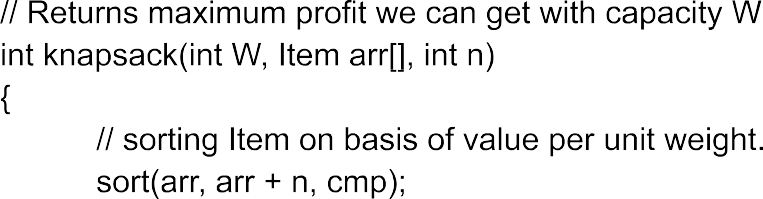




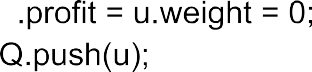


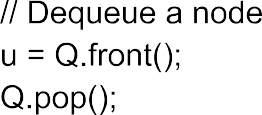


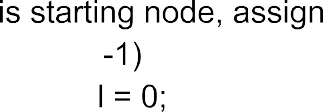










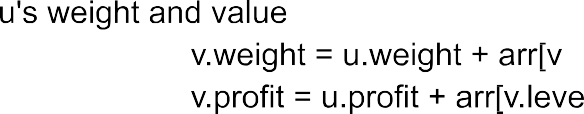




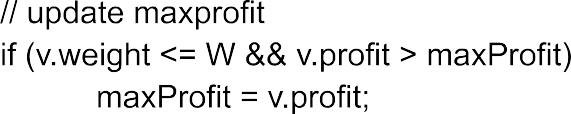






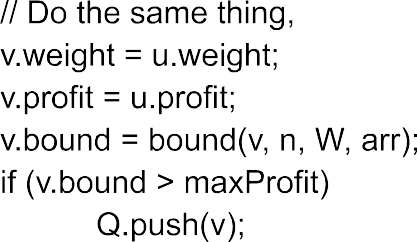




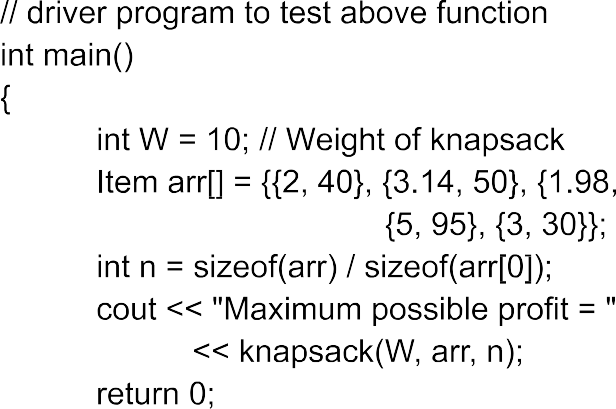










\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Assignment No: 6

**Title Name:** Design 8-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final 8-queen’s matrix.

**Name**: Abhishek Santosh Bankar

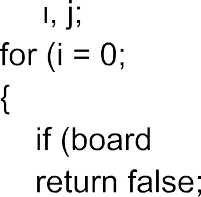
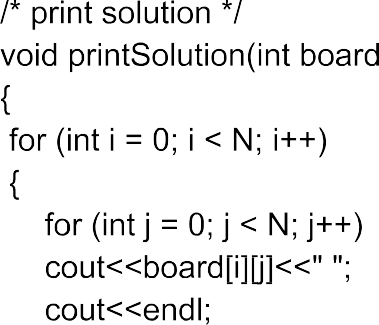
**Class** : BE **Div**: 1 **Batch**: A

## Roll No: 405A011

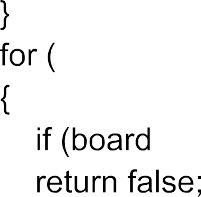
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Program:











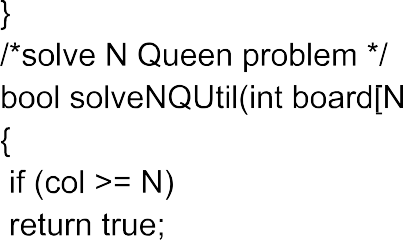




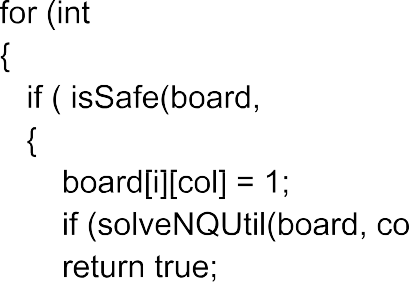




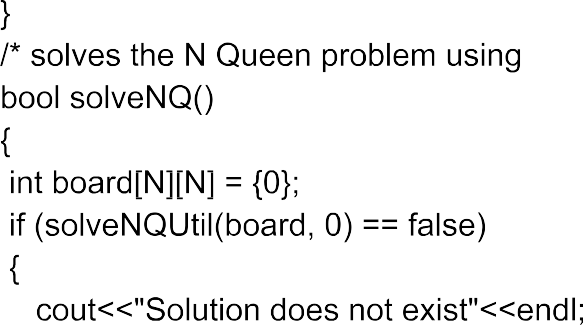




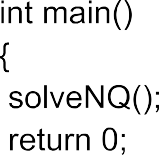


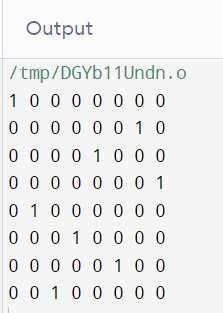












\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# DAA Mini Project

**Title Name:** Implement merge sort and multithreaded merge sort. Compare time required by both the algorithms. Also analyze the performance of each algorithm for the best case and the worst case.

**Name**: Abhishek Santosh Bankar

**Class** : BE **Div**: 1 **Batch**: A

# Roll No: 405A011

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Title**: Implement merge sort and multithreaded merge sort. Compare time required by both the algorithms. Also analyze the performance of each algorithm for the best case and the worst case.

**Problem Statement**: Write a program for merge sort and multithreaded merge sort and analyze the performance of each algorithm for the best case and the worst case.

**Prerequisites:** Design and analysis algorithm

**Objectives**: To understand for merge sort and multithreaded merge sort and analyze the performance of each algorithm for the best case and the worst case.

# Theory:

**Merge Sort:**

Merge sort is similar to the quick sort algorithm as it uses the divide and conquer approach to sort the elements. It is one of the most popular and efficient sorting algorithm. It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves. We have to define the merge() function to perform the merging.

The sub-lists are divided again and again into halves until the list cannot be divided further. Then we combine the pair of one element lists into two-element lists, sorting them in the process. The sorted two-element pairs is merged into the four-element lists, and so on until we get the sorted list.

Time Complexity

|  |  |
| --- | --- |
| **Case** | **Time Complexity** |
| **Best Case** | O(n\*logn) |
| **Average Case** | O(n\*logn) |
| **Worst Case** | O(n\*logn) |

## Multi-Threading:

In the operating system, **Threads** are the lightweight process which is responsible for executing the part of a task. Threads share common resources to execute the task concurrently.

**Multi-threading** is an implementation of multitasking where we can run multiple threads on a single processor to execute the tasks concurrently. It subdivides specific operations within a single application into individual threads. Each of the threads can run in parallel.

For Example-:

**In** −int arr[] = {3, 2, 1, 10, 8, 5, 7, 9, 4}

**Out** −Sorted array is: 1, 2, 3, 4, 5, 7, 8, 9, 10

## Program :





