CENTRAL UNIVERSITY OF RAJASTHAN

Department of Computer Science

NAME: AGHIL.M

ENROLLMENT NO. :2024MSCS021

PROGRAMME NAME: M.Sc. Computer Science.

COURSE NAME: Advance Algorithm Lab

COURSE CODE: CSC-408

SUBMITTED TO: Dr.Abhay Kumar Rai

INDEX

PROGRAM -1

Write a program to implement linear search algorithm

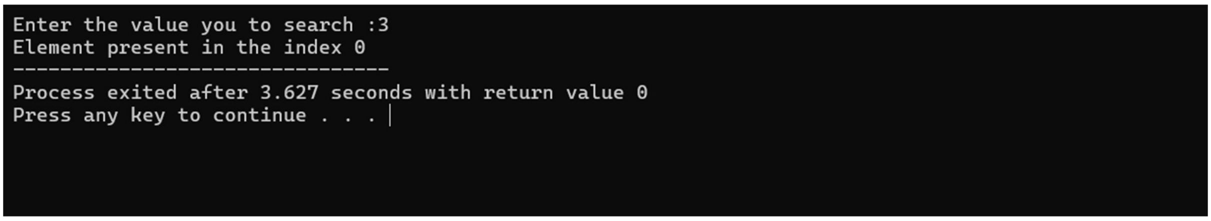
OBJECTIVE: To find an element in a data set by examining each element in the set, starting at the beginning and continuing until a match is found**.**

CODE:

```
#include <stdio.h>

#include <stdlib.h>

int linear_search(int a[],int size ,int search)

{

   int r =-2;

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

      if(search == a[i])

      {

         r=i;

         break;

      }

   return r;

}


int main()

{

   int a[]={3,10,24,59,98};

   int search;

   int size=sizeof(a)/sizeof(a[0]);
```

```
  printf("Enter the value you to search :");
  scanf("%d",&search);
  int result = linear_search(a,size,search);
  if(result == -2)
     printf("Element is not present in array");
  else
     printf("Element present in the index %d",result);
  return 0;
}
```

OUTPUT:



```
Enter the value you to search :3
Element present in the index 0
--------------------------------
Process exited after 3.627 seconds with return value 0
Press any key to continue . . . |
```

PROGRAM -2

Write a program to implement bubble sort without flag.

OBJECTIVE : To arrange a list of elements in order, such as from smallest to largest.

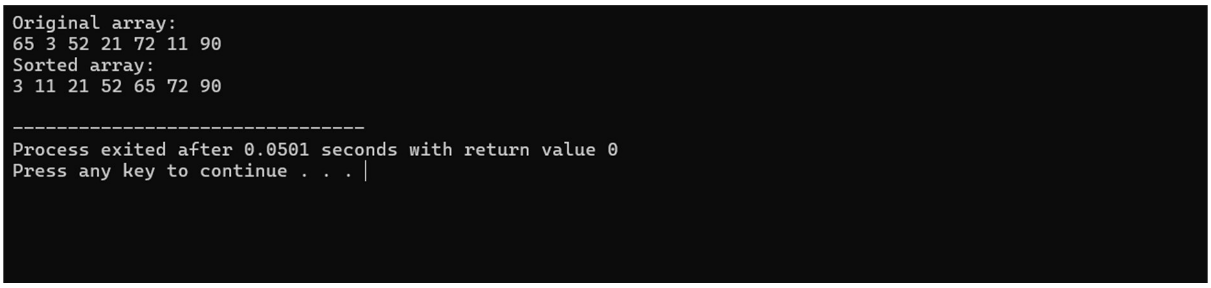
CODE:

```
#include <stdio.h>
#include <stdlib.h>
```

```
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
int main() {
    int arr[] = {65, 3, 52, 21, 72, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original array: \n");
    printArray(arr, n);
    bubbleSort(arr, n);
```

```
   printf("Sorted array: \n");

   printArray(arr, n);

   return 0;

}
```

OUTPUT:



```
Original array:
65 3 52 21 72 11 90
Sorted array:
3 11 21 52 65 72 90

--------------------------------
Process exited after 0.0501 seconds with return value 0
Press any key to continue . . . |
```

PROGRAM -3

Write a program to implement bubble sort with flag.

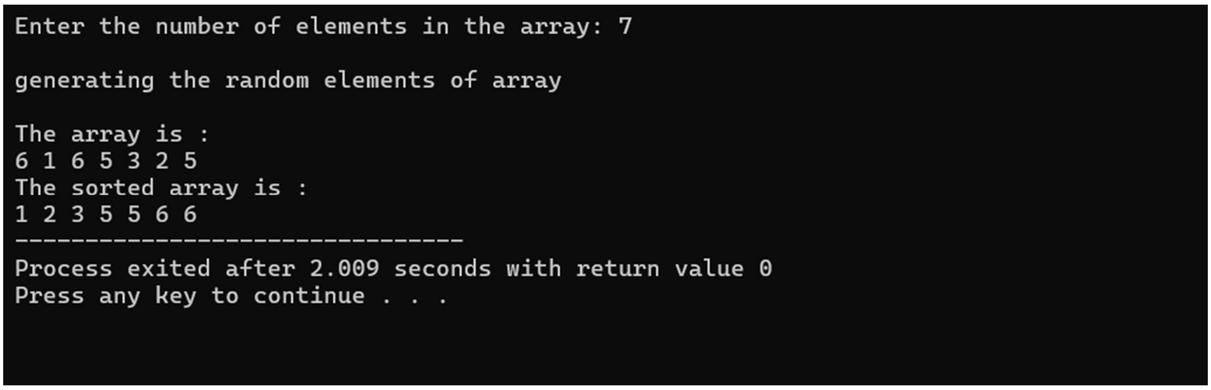
OBJECTIVE: To avoid unnecessary comparisons and reduce the time complexity of the algorithm.

CODE:

```
#include <stdio.h>

#include <stdlib.h>

int bubblesort(int a[],int n);

int main()


{

   int a[50],i,n;

   printf("Enter the number of elements in the array: ");
```

```
    scanf("%d",&n);
    printf("\ngenerating the random elements of array\n");
    for(i=0;i<n;i++)
    {
        a[i]= rand()%n;
    }
    printf("\nThe array is : \n");
    for(i=0;i<n;i++)
    {
        printf("%d ",a[i]);
    }

    bubblesort(a,n);
    printf("\nThe sorted array is : \n");
    for(i=0;i<n;i++)
    {
        printf("%d ",a[i]);
    }
}
int bubblesort(int a[],int n)
{
    int i,j,temp,flag;
    for(i=0;i<n-1;i++)
    {
        flag =0;
        for(j=0;j<n-1-i;j++)
        {
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
                flag =1;
            }
        }
        if(flag == 0)
        {
            return 0;
        }
    }
    return 0;
}
```

OUTPUT:



```
Enter the number of elements in the array: 7

generating the random elements of array

The array is :
6 1 6 5 3 2 5
The sorted array is :
1 2 3 5 5 6 6
--------------------------------
Process exited after 2.009 seconds with return value 0
Press any key to continue . . .
```

PROGRAM -4

Write a program to implement binary search algorithm.

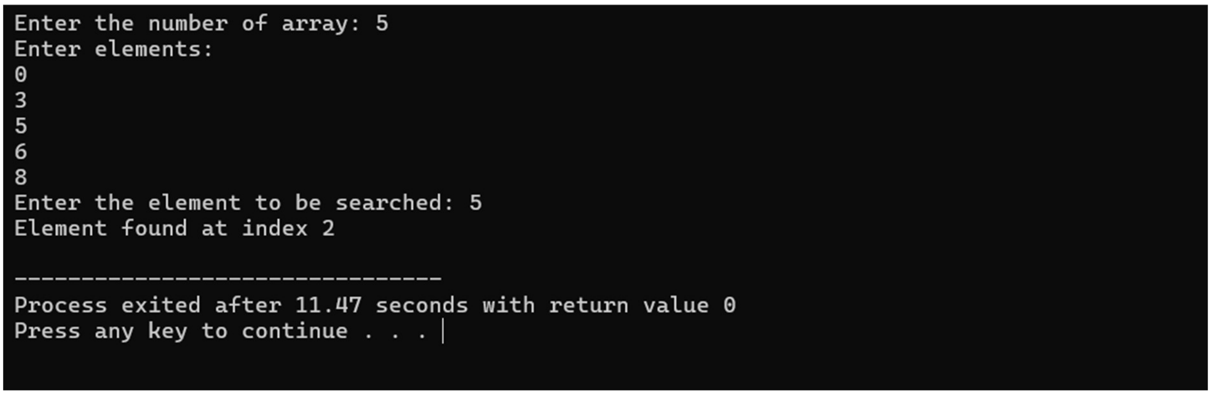
OBJECTIVE: To find the position of a target value in a sorted array.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
int binarySearch(int a[], int n, int x)
{
   int low = 0;
   int high = n - 1;
   while (low <= high)
   {
      int mid = (low + high) / 2;
      if (x < a[mid])
      {
         high = mid - 1;
      }
      else if (x > a[mid])
      {
         low = mid + 1;
      }
      else
      {
         return mid;
```

```
        }
    }
    return -1;
}


int main(){
    int a[50], n, x;
    printf("Enter the number of array: ");
    scanf("%d", &n);
    printf("Enter elements:\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }
    printf("Enter the element to be searched: ");
    scanf("%d", &x);
    int result = binarySearch(a, n, x);
    if (result == -1)
    {
        printf("Element not found\n");
    }
    else
    {
        printf("Element found at index %d\n", result);
    }
    return 0;
}
```

OUTPUT:



```
Enter the number of array: 5
Enter elements:
0
3
5
6
8
Enter the element to be searched: 5
Element found at index 2

--------------------------------
Process exited after 11.47 seconds with return value 0
Press any key to continue . . .
```

PROGRAM – 5

Write a program to implement merge sort algorithm.

OBJECTIVE: The continuously cuts down a list into multiple sublists until each has only one item, then merges those sublists into a sorted list.

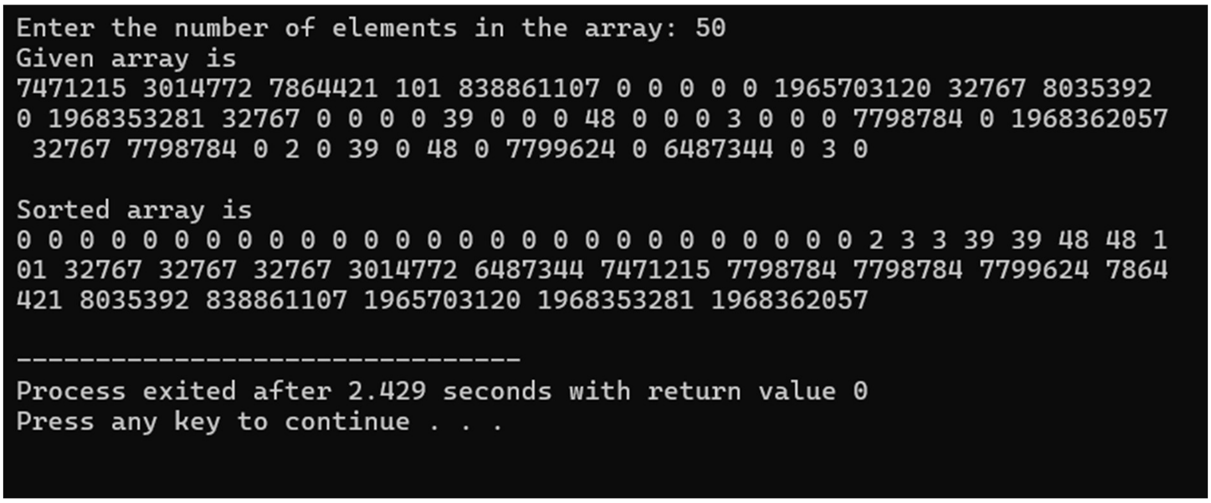
CODE:

```
#include <stdio.h>

#define MAX 100

int b[MAX];

void merge(int arr[], int low, int mid, int high) {

    int h = low;

    int i = low;

    int j = mid + 1;

    for (; h <= mid && j <= high; i++) {

        if (arr[h] <= arr[j]) {
            b[i] = arr[h];
            h++;
        } else {
            b[i] = arr[j];
            j++;
        }
    }
    for (; h <= mid; h++, i++) {
        b[i] = arr[h];
    }
    for (; j <= high; j++, i++) {
        b[i] = arr[j];
    }
    for (i = low; i <= high; i++) {
        arr[i] = b[i];
    }
}

void mergeSort(int arr[], int low, int high) {
    if (low < high) {
        int mid = (low + high) / 2;
        mergeSort(arr, low, mid);
        mergeSort(arr, mid + 1, high);
        merge(arr, low, mid, high);
    }
}
```

```
void printArray(int arr[], int n) {
        int i;
  for ( i = 0; i < n; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
}

int main() {
  int arr[MAX];
  int i,n;
  printf("Enter the number of elements in the array: ");
  scanf("%d", &n);
  FILE* file = fopen("../random_variables.txt", "r");
  if (file == NULL) {
    printf("Error opening file\n");
    return 1;
  }
  for ( i = 0; i < n; i++) {
    fscanf(file, "%d", &arr[i]);
  }
  fclose(file);
  printf("Given array is \n");
  printArray(arr, n);
  mergeSort(arr, 0, n - 1);
```

```
    printf("\nSorted array is \n");
    printArray(arr, n);
    return 0;
}
```

OUTPUT:



```
Enter the number of elements in the array: 50
Given array is
7471215 3014772 7864421 101 838861107 0 0 0 0 0 1965703120 32767 8035392
0 1968353281 32767 0 0 0 0 39 0 0 0 48 0 0 0 3 0 0 0 7798784 0 1968362057
 32767 7798784 0 2 0 39 0 48 0 7799624 0 6487344 0 3 0

Sorted array is
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 3 3 39 39 48 48 1
01 32767 32767 32767 3014772 6487344 7471215 7798784 7798784 7799624 7864
421 8035392 838861107 1965703120 1968353281 1968362057

--------------------------------
Process exited after 2.429 seconds with return value 0
Press any key to continue . . .
```

PROGRAM -6

Write a program to implement Quick sort algorithm.

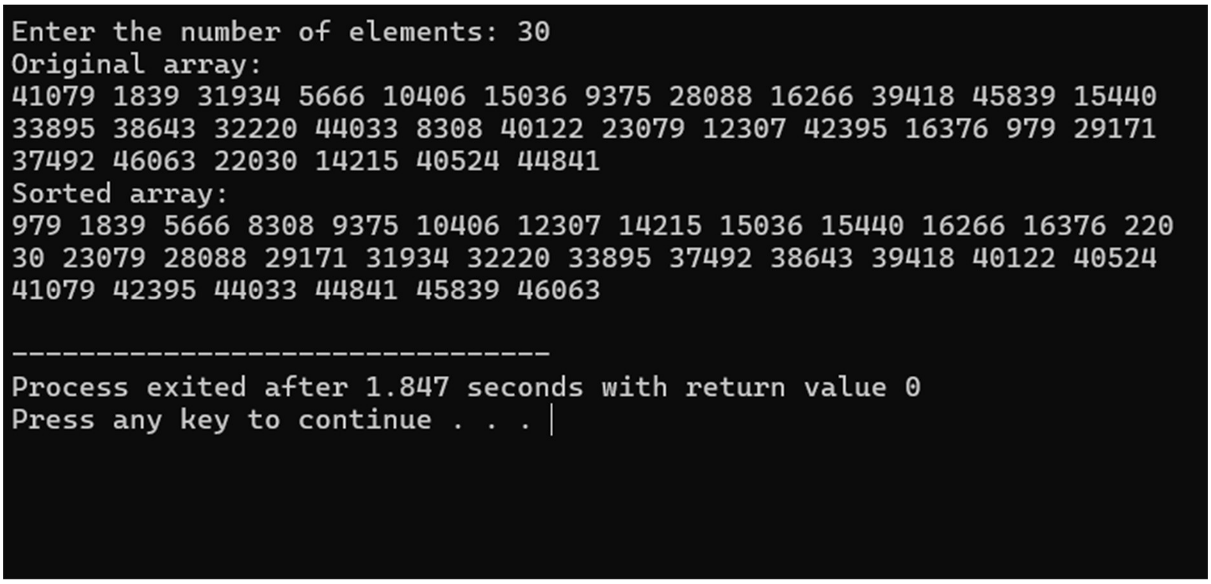
OBJECTIVE: To sort an array or list of elements using a divide-and-conquer strategy.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
```

```
}
void printArray(int arr[], int n) {
            int i;
    for ( i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    int j;

    for (j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }

    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivotIndex = partition(arr, low, high);
        quickSort(arr, low, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, high);
    }
}
int main() {
    int n,i;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    FILE* file = fopen("random_variables.txt", "r");
    if (file == NULL) {
        printf("Error opening file\n");
        return 1;
    }
    for ( i = 0; i < n; i++) {
        fscanf(file, "%d", &arr[i]);
    }
    fclose(file);
    printf("Original array: \n");
    printArray(arr, n);
    quickSort(arr, 0, n - 1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

OUTPUT:



```
Enter the number of elements: 30
Original array:
41079 1839 31934 5666 10406 15036 9375 28088 16266 39418 45839 15440
33895 38643 32220 44033 8308 40122 23079 12307 42395 16376 979 29171
37492 46063 22030 14215 40524 44841
Sorted array:
979 1839 5666 8308 9375 10406 12307 14215 15036 15440 16266 16376 220
30 23079 28088 29171 31934 32220 33895 37492 38643 39418 40122 40524
41079 42395 44033 44841 45839 46063

--------------------------------
Process exited after 1.847 seconds with return value 0
Press any key to continue . . .
```

PROGRAM -7

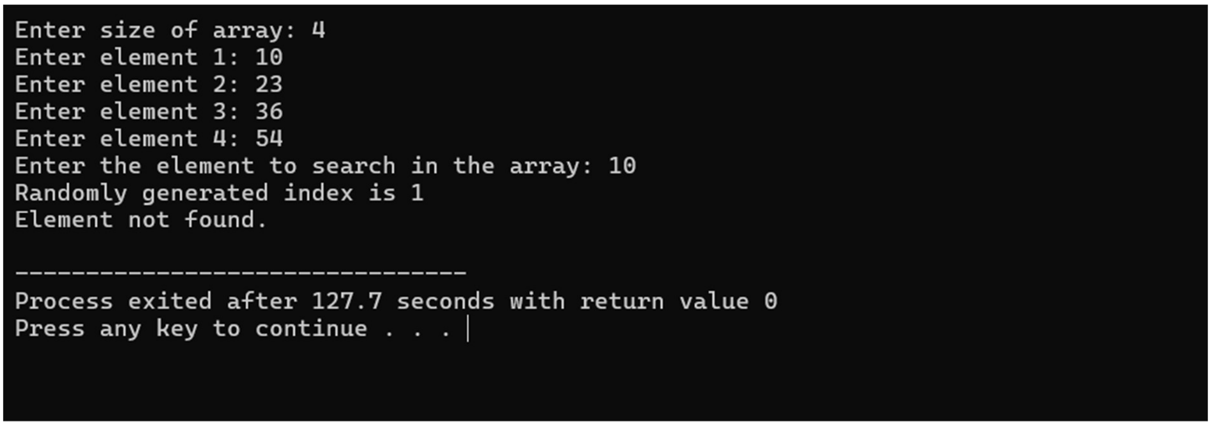
Write a program to implement non deterministic linear search algorithm

CODE:

```
#include <stdio.h>

#include <stdlib.h>

int ndlinear(int arr[], int size, int x) {

   int res = -1;

   int j = rand() % size;

   printf("Randomly generated index is %d\n", j);

   if (arr[j] == x)
```

```
        return j + 1;
    return res;
}

int main() {
    int i, n, x, result;
    printf("Enter size of array: ");
    scanf("%d", &n);
    int a[n];
    for (i = 0; i < n; i++) {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &a[i]);
    }
    printf("Enter the element to search in the array: ");
    scanf("%d", &x);
    result = ndlinear(a, n, x);
    if (result != -1)
        printf("Element %d found at %d position.\n", x, result);
    else
        printf("Element not found.\n");

    return 0;
}
```

OUTPUT:



```
Enter size of array: 4
Enter element 1: 10
Enter element 2: 23
Enter element 3: 36
Enter element 4: 54
Enter the element to search in the array: 10
Randomly generated index is 1
Element not found.

--------------------------------
Process exited after 127.7 seconds with return value 0
Press any key to continue . . .
```

PROGRAM -8

COUNTING SORT

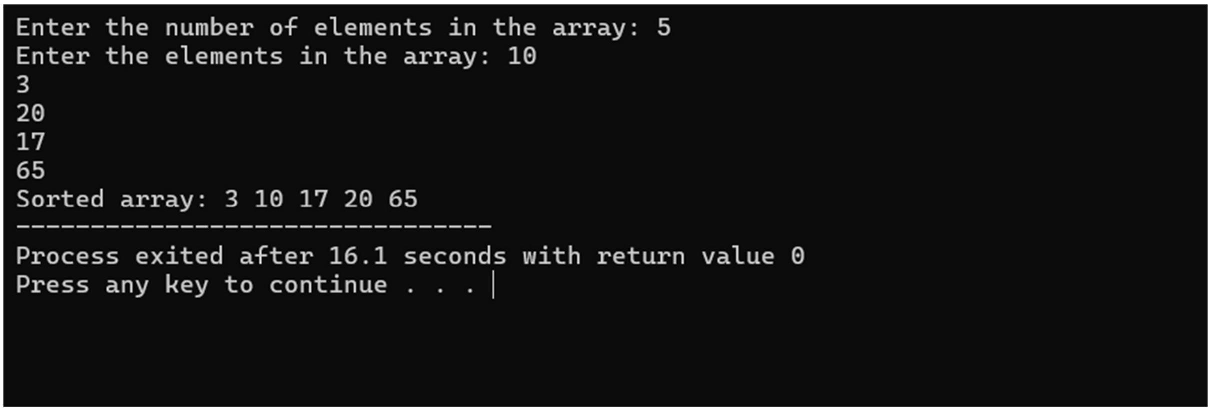
OBJECTIVE : The aim of the counting sort algorithm is to sort a collection of objects based on keys that are small positive integers.

CODE:

```
#include <stdio.h>
int findMax(int A[], int n) {
   int max = A[0];
   for (int i = 1; i < n; i++) {
      if (A[i] > max) {
         max = A[i];
      }
   }
   return max;
}
```

```
 void countingSort(int A[], int n, int k) {
    int B[n];
    int C[k + 1];
    for (int i = 0; i <= k; i++) {
        C[i] = 0;
    }
    for (int j = 0; j < n; j++) {
        C[A[j]]++;
    }
    for (int i = 1; i <= k; i++) {
        C[i] += C[i - 1];
    }
    for (int j = n - 1; j >= 0; j--) {
        B[C[A[j]] - 1] = A[j];
        C[A[j]]--;
    }
    for (int i = 0; i < n; i++) {
        A[i] = B[i];
    }
}

int main() {
    int n;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);
    int A[n];
    printf("Enter the elements in the array: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &A[i]);
    }
    int k = findMax(A, n);
    countingSort(A, n, k);
printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", A[i]);
    }
    return 0;
}
```

OUTPUT:



```
Enter the number of elements in the array: 5
Enter the elements in the array: 10
3
20
17
65
Sorted array: 3 10 17 20 65
--------------------------------
Process exited after 16.1 seconds with return value 0
Press any key to continue . . .
```

PROGRAM -9

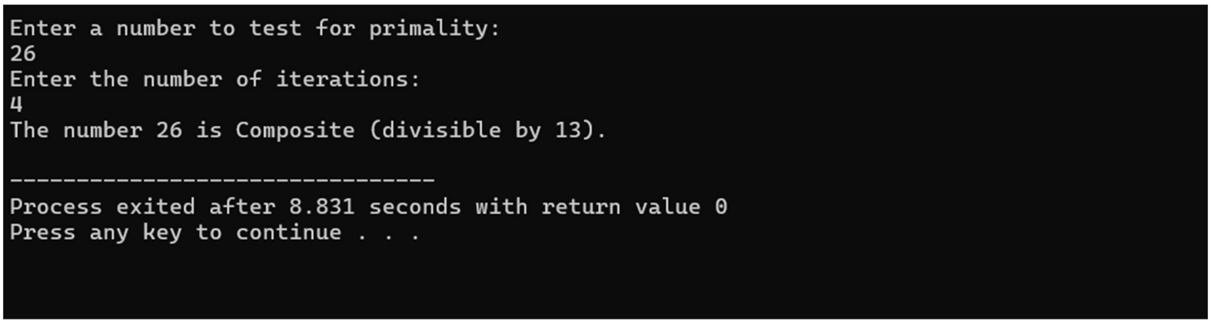
WAP Non-Deterministic Primality checking

OBJECTIVE: To determine if a number is prime with a probabilistic method primarily.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
int ptesting(int n, int k) {
    int i = 1, remainder, r;
    if (n <= 1) {
        printf("The number %d is neither prime nor composite.\n", n);
        return 0;
    }

L:
    r = (rand() % (n - 2)) + 2;
    remainder = n % r;
    if (remainder == 0)
        goto out;
    else
        i++;

    if (i <= k)
        goto L;
    printf("The number %d is Prime.\n", n);
    return 0;

out:
    printf("The number %d is Composite (divisible by %d).\n", n, r);
    return 0;
}
int main() {
    int N, k;
    printf("Enter a number to test for primality:\n");
    scanf("%d", &N);
    printf("Enter the number of iterations:\n");
    scanf("%d", &k);
    ptesting(N, k);
    return 0;
}
```

OUTPUT:



```
Enter a number to test for primality:
26
Enter the number of iterations:
4
The number 26 is Composite (divisible by 13).

--------------------------------
Process exited after 8.831 seconds with return value 0
Press any key to continue . . .
```

PROGRAM -10

Write a program to implement non deterministic knapsack algorithm

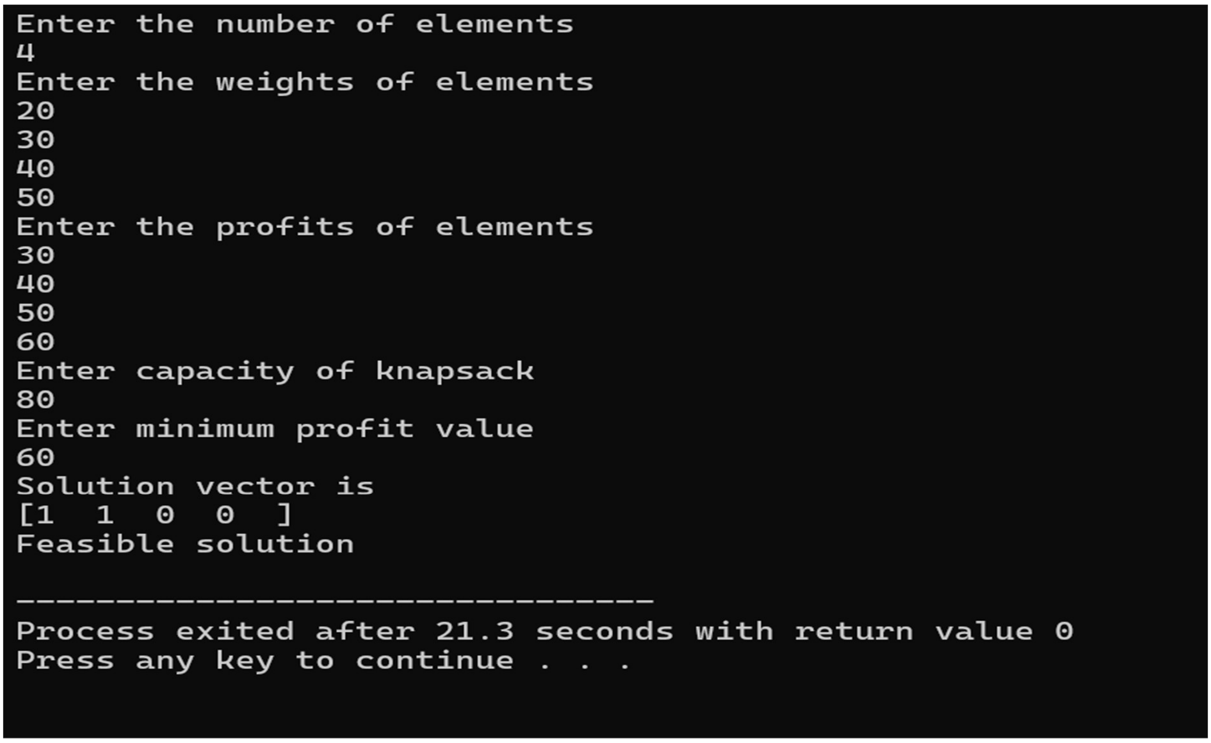
OBJECTIVE: To determine if a profit of at least a certain amount can be earned while staying within the knapsack's maximum capacity.

CODE:

```
#include<stdio.h>
#include<stdlib.h>
int main() {
   int n, M, k, i;
   int profit = 0, weight = 0;

   printf("Enter the number of elements \n");
   scanf("%d", &n);
   int w[n], p[n], x[n];

   printf("Enter the weights of elements\n");
   for (i = 0; i < n; i++)
      scanf("%d", &w[i]);

   printf("Enter the profits of elements\n");
   for (i = 0; i < n; i++)
```

```
        scanf("%d", &p[i]);


    printf("Enter capacity of knapsack\n");
    scanf("%d", &M);


    printf("Enter minimum profit value\n");
    scanf("%d", &k);
    for (i = 0; i < n; i++)
        x[i] = rand() % 2;
    printf("Solution vector is \n[");
    for (i = 0; i < n; i++)
        printf("%d  ", x[i]);
    printf("]\n");
    for (i = 0; i < n; i++) {
        profit += x[i] * p[i];
        weight += x[i] * w[i];
    }
    if (weight > M || profit < k)
        printf("Not feasible solution\n");
    else
        printf("Feasible solution\n");
    return 0;
}
```

OUTPUT:



```
Enter the number of elements
4
Enter the weights of elements
20
30
40
50
Enter the profits of elements
30
40
50
60
Enter capacity of knapsack
80
Enter minimum profit value
60
Solution vector is
[1  1  0  0  ]
Feasible solution

--------------------------------
Process exited after 21.3 seconds with return value 0
Press any key to continue . . .
```

PROGRAM-11

FRACTIONAL KNAPSACK USING GREEDY ALGORITHM

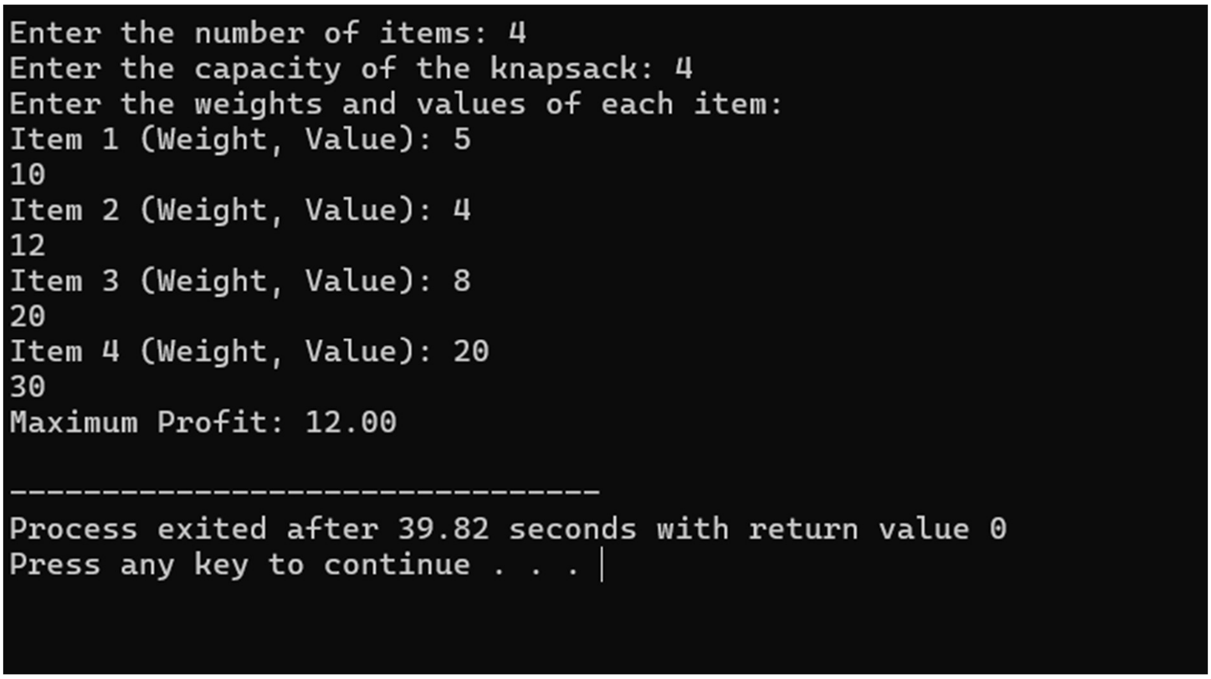
OBJECTIVE: To maximize the total value of items placed in a knapsack while staying within the knapsack's weight capacity.

CODE :

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

```
void fractionalKnapsack(int W[], int V[], int M, int n) {
    float cost[n], total = 0.0;
    int i, j;
    for (i = 0; i < n; i++) {
        cost[i] = (float)V[i] / W[i];
    }
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (cost[j] < cost[j + 1]) {
                float temp = cost[j];
                cost[j] = cost[j + 1];
                cost[j + 1] = temp;
                int tempW = W[j];
                W[j] = W[j + 1];
                W[j + 1] = tempW;
                int tempV = V[j];
                V[j] = V[j + 1];
                V[j + 1] = tempV;
            }
        }
    }
    i = 0;
    while (i < n) {
        if (W[i] <= M) {
            M -= W[i];
            total += V[i];
        } else {
            total += (float)V[i] * M / W[i];
            break;
        }
        i++;
}
    printf("Maximum Profit: %.2f\n", total);
}
int main() {
    int n, M;
    printf("Enter the number of items: ");
    scanf("%d", &n);
    printf("Enter the capacity of the knapsack: ");
    scanf("%d", &M);
    int W[n], V[n];
    printf("Enter the weights and values of each item:\n");
    for (int i = 0; i < n; i++) {
        printf("Item %d (Weight, Value): ", i + 1);
        scanf("%d %d", &W[i], &V[i]);
    }
    fractionalKnapsack(W, V, M, n);
            return 0;
}
```

OUTPUT:



```
Enter the number of items: 4
Enter the capacity of the knapsack: 4
Enter the weights and values of each item:
Item 1 (Weight, Value): 5
10
Item 2 (Weight, Value): 4
12
Item 3 (Weight, Value): 8
20
Item 4 (Weight, Value): 20
30
Maximum Profit: 12.00

--------------------------------
Process exited after 39.82 seconds with return value 0
Press any key to continue . . .
```

PROGRAM -12

8 – QUEENS PROBLEM

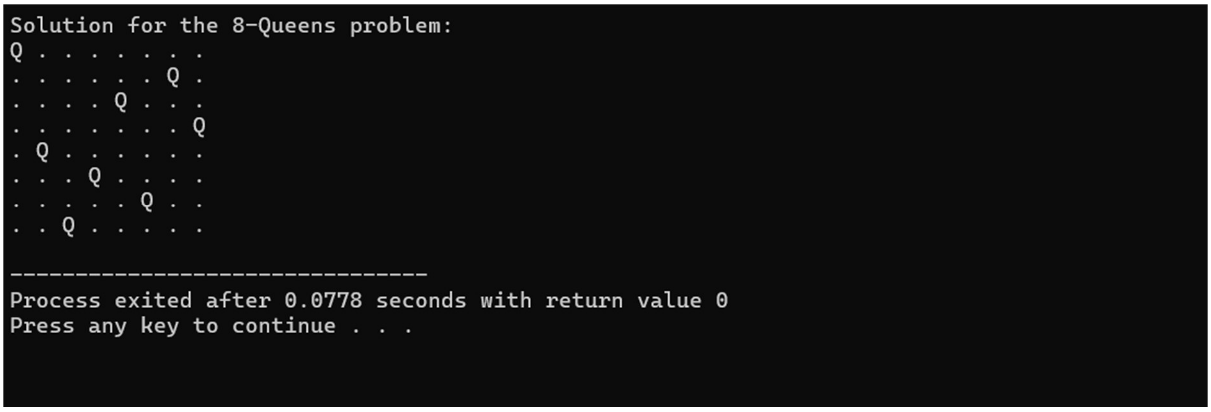
OBJECTIVE: The aim of the 8 Queens Problem using backtracking is to place eight queens on an 8x8 chessboard so that no two queens threaten each other.

CODE:

```
#include <stdio.h>

#include <stdbool.h>

#define SIZE 8
```

```
bool isSafe(int board[SIZE][SIZE], int row, int col) {
    int i, j;
    for (i = 0; i < col; i++) {
        if (board[row][i]) {
            return false;
        }
    }
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j]) {
            return false;
        }
    }
    for (i = row, j = col; i < SIZE && j >= 0; i++, j--) {
        if (board[i][j]) {
            return false;
        }
    }
    return true;
}
bool solve8QueensUtil(int board[SIZE][SIZE], int col) {
    if (col >= SIZE) {
        return true;
    }
    for (int i = 0; i < SIZE; i++) {
        if (isSafe(board, i, col)) {
            board[i][col] = 1;
            if (solve8QueensUtil(board, col + 1)) {
                return true;
            }
            board[i][col] = 0;
        }
    }
   return false;
}
void solve8Queens() {
    int board[SIZE][SIZE] = {0};
           if (solve8QueensUtil(board, 0)) {
        printf("Solution for the 8-Queens problem:\n");
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                printf("%c ", board[i][j] ? 'Q' : '.');
            }
            printf("\n");
        }
    } else {
        printf("No solution exists for the 8-Queens problem.\n");
    }
}
int main() {
    solve8Queens();
    return 0;
}
```

OUTPUT :



```
Solution for the 8-Queens problem:
Q . . . . . . .
. . . . . . Q .
. . . . Q . . .
. . . . . . . Q
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
. . Q . . . . .

--------------------------------
Process exited after 0.0778 seconds with return value 0
Press any key to continue . . .
```

PROGRAM-13

RANDOMIZED QUICK SORT

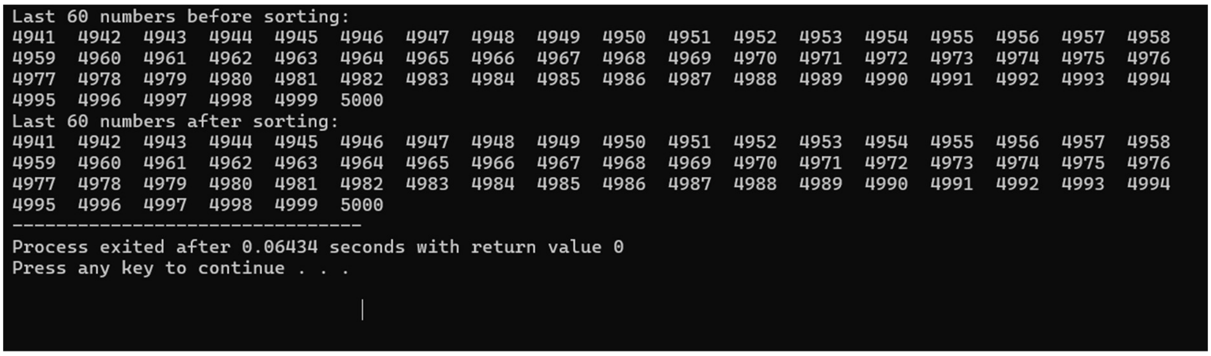
CODE :

```
#include<stdio.h>
#include <stdlib.h>
#include <time.h>
#define NUM_NUMBERS 5000
void interchange(int a[], int i, int j)
{
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
```

```
}
int partition(int a[], int low, int high) {
        int random_index = low + rand() % (high - low);
        interchange(a, low, random_index);
        int pivot = a[low], i = low, j = high;
        do
        {
                do
                {
                        i++;
                } while (a[i] < pivot && i <= high);
                do
                {
                        j--;
                } while (a[j] > pivot && j >= low);
                if (i < j)
                {
                        interchange(a, i, j);
                }
        } while (i < j);
        interchange(a, low, j);
        return j;
}

void quicksort(int a[], int low, int high)
{
        if (low < high)
        {
                int j = partition(a, low, high + 1);
                quicksort(a, low, j - 1);
                quicksort(a, j + 1, high);
        }
}
int main()
{
        int i;
        int *arr = (int *)malloc(NUM_NUMBERS * sizeof(int));
        srand(time(0));
        for (i = 0; i < NUM_NUMBERS; i++)
        {
                arr[i] = i + 1;
        }
        printf("Last 60 numbers before sorting:\n");
        for (i = NUM_NUMBERS - 60; i < NUM_NUMBERS; i++)
        {
                printf("%d  ", arr[i]);
   }
        printf("\n\n");
        quicksort(arr, 0, NUM_NUMBERS - 1);
        printf("Last 60 numbers after sorting:\n");
        for (i = NUM_NUMBERS - 60; i < NUM_NUMBERS; i++)
        {
```

```
        printf("%d  ", arr[i]);
    }
    printf("\n");
    free(arr);
    return 0;
}
```

OUTPUT :



```
Last 60 numbers before sorting:
4941  4942  4943  4944  4945  4946  4947  4948  4949  4950  4951  4952  4953  4954  4955  4956  4957  4958
4959  4960  4961  4962  4963  4964  4965  4966  4967  4968  4969  4970  4971  4972  4973  4974  4975  4976
4977  4978  4979  4980  4981  4982  4983  4984  4985  4986  4987  4988  4989  4990  4991  4992  4993  4994
4995  4996  4997  4998  4999  5000
Last 60 numbers after sorting:
4941  4942  4943  4944  4945  4946  4947  4948  4949  4950  4951  4952  4953  4954  4955  4956  4957  4958
4959  4960  4961  4962  4963  4964  4965  4966  4967  4968  4969  4970  4971  4972  4973  4974  4975  4976
4977  4978  4979  4980  4981  4982  4983  4984  4985  4986  4987  4988  4989  4990  4991  4992  4993  4994
4995  4996  4997  4998  4999  5000
--------------------------------
Process exited after 0.06434 seconds with return value 0
Press any key to continue . . .
```

PROGRAM-14

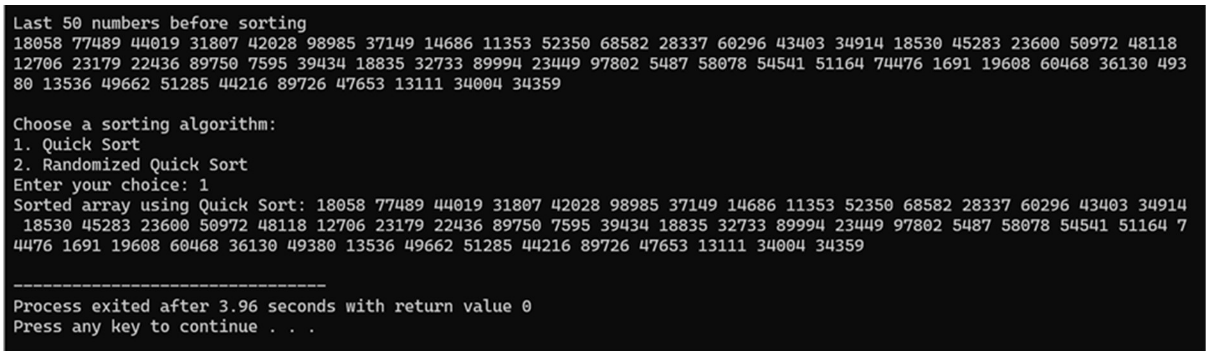
Comparison Between Non-Randomized and Randomized Quick Sort in Worst Case

CODE :

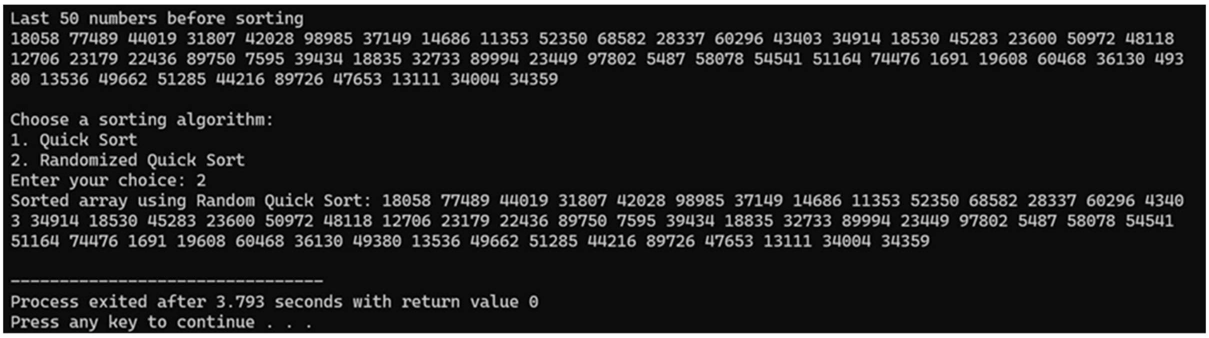
```
//Header File
#ifndef quick_H
#define quick_H
void interchange(int a[],int i,int j);
int partition(int a[], int low, int high);
int randompartition(int a[], int low, int high);
void quicksort(int a[],int low,int high);
void rquicksort(int a[],int low,int high);
#endif
//Implementation File
#include<stdio.h>
#include<stdlib.h>
#include "quick_H.h"
void interchange(int a[],int i,int j)
{
        int p=a[i];
        a[i]=a[j];
        a[j]=p;
}
int partition(int a[],int low,int high)
{
        int pivot=a[low],i=low,j=high;
        do
        {
                do
                {
                        i++;
                }while(a[i]<pivot && (i<=high));
                do
                {
                        j--;
                }while(a[j]>pivot && (j>=low));
                if(i<j)
        {
            interchange(a,i,j);
        }
    }while(i<j);
    a[low]=a[j];
    a[j]=pivot;
    return j;
}
int randompartition(int a[], int low, int high)
{
    int random_index = low + rand() % (high - low);
    interchange(a, low, random_index);
    int pivot = a[low], i = low, j = high;
    do
    {
    do
        {
            i++;
        } while (a[i] < pivot && i <= high);
        do
        {
            j--;
        } while (a[j] > pivot && j >= low);
        if (i < j)
        {
            interchange(a, i, j);
        }
    } while (i < j);
    interchange(a, low, j);
    return j;
}
void quicksort(int a[],int low,int high)
{
    int j;
    if(low<high)
    {
        j=partition(a,low,high+1);
        quicksort(a,low,j-1);
        quicksort(a,j+1,high);
    }
}
void rquicksort(int a[], int low, int high)
{
    if (low < high)
    {
        int j = randompartition(a, low, high + 1);
        rquicksort(a, low, j - 1);
        rquicksort(a, j + 1, high);
    }
}
//Driver Code
#include<stdio.h>
```

```
#include<stdlib.h>
#include "quick_H.h"
#define NUM_NUMBERS 10000000
void print_array(int arr[], int n) {
}
    for (int i = n-50; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
int main() {
    int i;
            int *arr=(int *)malloc(NUM_NUMBERS *sizeof(int));
            const char *file_path="C:\\Users\\Redmi\\OneDrive\\Documents\\VIIth\\Advanced Algorithms\\AA Lab\\random_number.txt ";
            FILE *file=fopen(file_path,"r");
            for(i=0;i<NUM_NUMBERS;i++)
            {
                        if(fscanf(file,"%d",&arr[i])!=1)
                        {
                                    printf("Error\n");
                                    fclose(file);
                        }
            }
    print_array(arr,NUM_NUMBERS);
    int choice;
    printf("\nChoose a sorting algorithm:\n");
    printf("1. Quick Sort\n");
    printf("2. Randomized Quick Sort\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice) {
        case 1:
            void quicksort(int a[],int low,int high);
            printf("Sorted array using Quick Sort: ");
            break;
        case 2:
            void rquicksort(int a[], int low, int high);
            printf("Sorted array using Random Quick Sort: ");
            break;
        default:
            printf("Invalid choice.\n");
            return 1;
    }
    print_array(arr, NUM_NUMBERS);
    return 0;
}
```

OUTPUT:



```
Last 50 numbers before sorting
18058 77489 44019 31807 42028 98985 37149 14686 11353 52350 68582 28337 60296 43403 34914 18530 45283 23600 50972 48118
12706 23179 22436 89750 7595 39434 18835 32733 89994 23449 97802 5487 58078 54541 51164 74476 1691 19608 60468 36130 493
80 13536 49662 51285 44216 89726 47653 13111 34004 34359

Choose a sorting algorithm:
1. Quick Sort
2. Randomized Quick Sort
Enter your choice: 1
Sorted array using Quick Sort: 18058 77489 44019 31807 42028 98985 37149 14686 11353 52350 68582 28337 60296 43403 34914
 18530 45283 23600 50972 48118 12706 23179 22436 89750 7595 39434 18835 32733 89994 23449 97802 5487 58078 54541 51164 7
4476 1691 19608 60468 36130 49380 13536 49662 51285 44216 89726 47653 13111 34004 34359

--------------------------------
Process exited after 3.96 seconds with return value 0
Press any key to continue . . .
```



```
Last 50 numbers before sorting
18058 77489 44019 31807 42028 98985 37149 14686 11353 52350 68582 28337 60296 43403 34914 18530 45283 23600 50972 48118
12706 23179 22436 89750 7595 39434 18835 32733 89994 23449 97802 5487 58078 54541 51164 74476 1691 19608 60468 36130 493
80 13536 49662 51285 44216 89726 47653 13111 34004 34359

Choose a sorting algorithm:
1. Quick Sort
2. Randomized Quick Sort
Enter your choice: 2
Sorted array using Random Quick Sort: 18058 77489 44019 31807 42028 98985 37149 14686 11353 52350 68582 28337 60296 4340
3 34914 18530 45283 23600 50972 48118 12706 23179 22436 89750 7595 39434 18835 32733 89994 23449 97802 5487 58078 54541
51164 74476 1691 19608 60468 36130 49380 13536 49662 51285 44216 89726 47653 13111 34004 34359

--------------------------------
Process exited after 3.793 seconds with return value 0
Press any key to continue . . .
```

PROGRAM – 15

SUM OF SUBSETS USING BACKTRACKING

OBJECTIVE: The goal of the subset sum algorithm is to determine if a subset of a given set of integers adds up to a target sum.

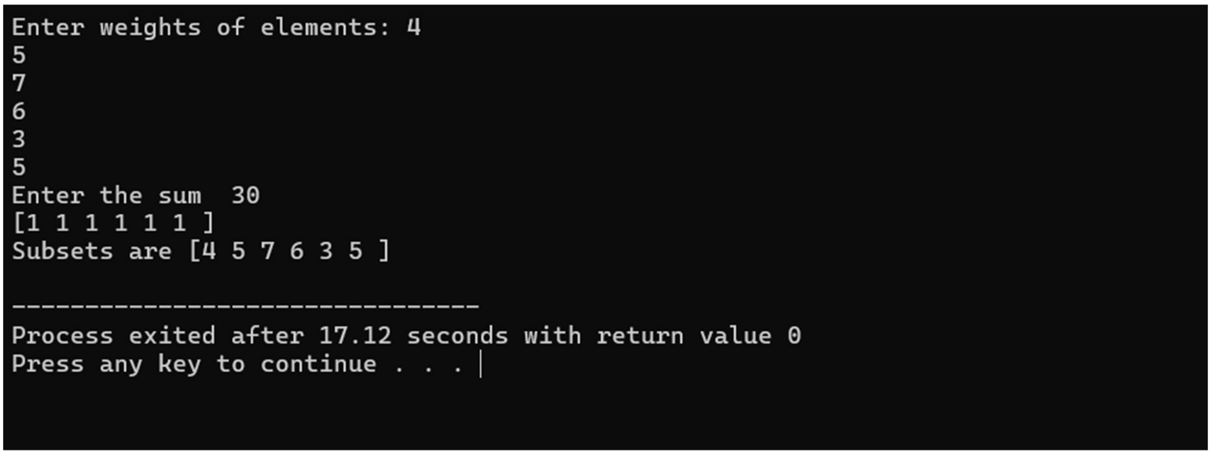
CODE :

```
#include<stdio.h>

#include<stdlib.h>

int w[6],x[6]={0},m,r=0,n=6;

void print(int x[],int n)

{

int i;
```

```
printf("[");
for(i=0;i<n;i++)
    printf("%d ",x[i]);
    printf("]\n");
    printf("Subsets are [");
     for(int i=0;i<6;i++)
    {
     if(x[i]==1)
     {
        printf("%d ",w[i]);
     }
    }
    printf("]\n");


}
void sumofsubset(int s,int k,int r)
{
int i;
if(k>=n)
return ;
    x[k]=1;
if(s+w[k]==m)
{
print(x,n);
}
```

```
else if( k+1<n &&  s+w[k]+w[k+1]<=m)
sumofsubset(s+w[k],k+1,r-w[k]);
x[k]=0;
if(s+r-w[k]>=m && k+1 < n &&  s+w[k+1]<=m)
{
sumofsubset(s,k+1,r-w[k]);
}
}
int main()
{
printf("Enter weights of elements: ");
   for(int i=0;i<6;i++)
   {
      scanf("%d",&w[i]);
      r+=w[i];
   }
   printf("Enter the sum  ");
   scanf("%d",&m);
   //print(w,n);
   sumofsubset(0,0,r);
}
```

OUTPUT :



```
Enter weights of elements: 4
5
7
6
3
5
Enter the sum  30
[1 1 1 1 1 1 ]
Subsets are [4 5 7 6 3 5 ]

--------------------------------
Process exited after 17.12 seconds with return value 0
Press any key to continue . . .
```

PROGRAM-16

COMPARISON BETWEEN Bubble Sort(without ,with flag),Quick Sort,Merge Sort

CODE :

```
//Header File

#ifndef sortings_H

#define sortings_H

void merge(int a[],int low,int mid,int high);

void mergesort(int A[],int low,int high);

void interchange(int a[],int i,int j);

int partition(int a[],int low,int high);


void quicksort(int a[],int low,int high);
void bubbleSort_noflag(int arr[], int n);
void bubbleSort_flag(int arr[], int n);
#endif

//Implementation File
#include<stdio.h>
#include <stdlib.h>
#include "sortings_H.h"
#define NUM_NUMBERS 10000
void merge(int a[],int low,int mid,int high)
{
    int *b=(int *)malloc(NUM_NUMBERS *sizeof(int));
    if(b == NULL)
    printf("Memory not allocated");
    int k=low,i=low,j=mid+1;
    while(i <= mid && j <= high)
    {
        if(a[i] <= a[j])
        {
            b[k++]=a[i++];
        }
        else
            b[k++]=a[j++];
    }

    while(i <= mid)
```

```
        b[k++]=a[i++];
        while(j <= high)
        b[k++]=a[j++];
        for(i = low;i <= high; i++)
         a[i]=b[i];
        free(b);
}
void mergesort(int A[],int low,int high)
{
        if(low < high)
        {
                int mid=(low + high) / 2;
                mergesort(A,low,mid);
                mergesort(A,mid + 1,high);
                merge(A,low,mid,high);
        }
}
void interchange(int a[],int i,int j)
{
        int p=a[i];
        a[i]=a[j];
        a[j]=p;
}
int partition(int a[],int low,int high)
{
        int pivot=a[low],i=low,j=high;
```

```
    do
    {
        do
        {
            i++;
        }while(a[i]<pivot && (i<=high));
        do
        {
            j--;
        }while(a[j]>pivot && (j>=low));
        if(i<j)
        {
            interchange(a,i,j);
        }
    }while(i<j);
    a[low]=a[j];
    a[j]=pivot;
    return j;
}
void quicksort(int a[],int low,int high)
{
    int j;
    if(low<high)
    {
        j=partition(a,low,high+1);
        quicksort(a,low,j-1);
```

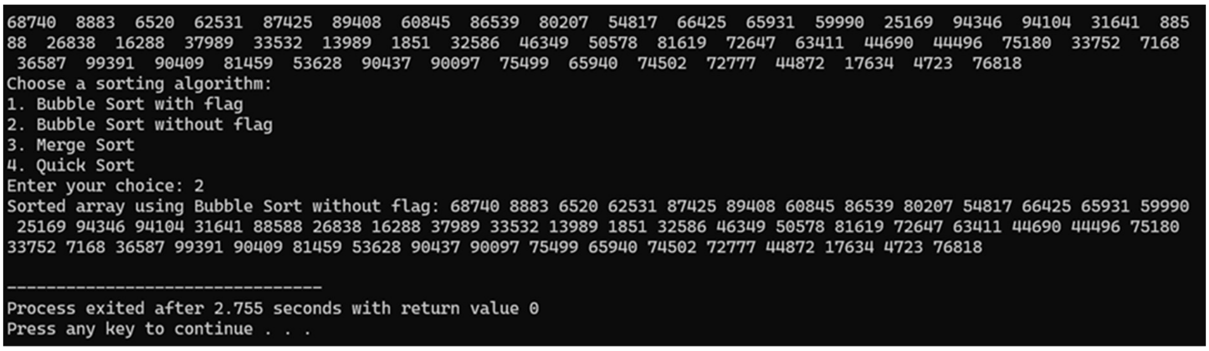
```
                        quicksort(a,j+1,high);
            }
}
void bubbleSort_noflag(int arr[], int n) {
            int i,j;
    for (i = 0; i < NUM_NUMBERS-1; i++) {
        for (j = 0; j < NUM_NUMBERS- i-1 ; j++) {
            if (arr[j] > arr[j + 1])
                interchange(arr, j, j + 1);
        }
    }
}
void bubbleSort_flag(int arr[], int n) {
            int i,j,flag;
    for (i = 0; i < NUM_NUMBERS-1; i++) {
            flag=0;
        for (j = 0; j < NUM_NUMBERS- i-1 ; j++) {
            if (arr[j] > arr[j + 1])
                {
                        interchange(arr, j, j + 1);
                    flag=1;
                }
        }
        if (flag == 0)
            break;
                        }
```

```
}
//Driver Code
#include <stdio.h>
#include <stdlib.h>
#include "sortings_H.h"
#define NUM_NUMBERS 1000000
void print_array(int arr[], int n) {
        int i;
   for (i = n-50; i < n; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");
}
int main()
{
        int i;
        int *arr=(int *)malloc(NUM_NUMBERS *sizeof(int));
        const char *file_path="D:\\7th semester\\rand_nm.txt";
        FILE *file=fopen(file_path,"r");
        for(i=0;i<NUM_NUMBERS;i++)
        {
                if(fscanf(file,"%d",&arr[i])!=1)
                {
                        printf("Error\n");
                        fclose(file);
                }
```

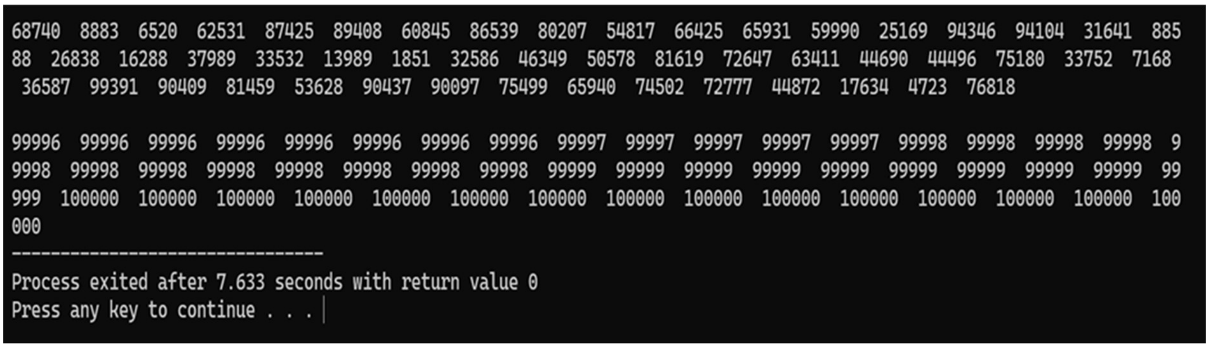
```
    }
    for(i=NUM_NUMBERS-50;i<NUM_NUMBERS;i++)
    {
          printf("%d  ",arr[i]);
    }
int choice;
printf("\nChoose a sorting algorithm:\n");
printf("1. Bubble Sort with flag\n");
printf("2. Bubble Sort without flag\n");
printf("3. Merge Sort\n");
printf("4. Quick Sort\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
  case 1:
    bubbleSort_flag(arr,NUM_NUMBERS);
    printf("Sorted array using Bubble Sort with flag: ");
    break;
  case 2:
    bubbleSort_noflag(arr,NUM_NUMBERS);
    printf("Sorted array using Bubble Sort without flag: ");
    break;
  case 3:
    mergesort(arr,0,NUM_NUMBERS-1);
    printf("Sorted array using Merge Sort: ");
    break;
```

```
        case 4:
            quicksort(arr,0,NUM_NUMBERS-1);
            printf("Sorted array using Quick Sort: ");
            break;
        default:
            printf("Invalid choice.\n");
            return 1;
    }
    print_array(arr,NUM_NUMBERS);
    return 0;
}
```

OUTPUT



```
68740  8883  6520  62531  87425  89408  60845  86539  80207  54817  66425  65931  59990  25169  94346  94104  31641  885
88  26838  16288  37989  33532  13989  1851  32586  46349  50578  81619  72647  63411  44690  44496  75180  33752  7168
 36587  99391  90409  81459  53628  90437  90097  75499  65940  74502  72777  44872  17634  4723  76818
Choose a sorting algorithm:
1. Bubble Sort with flag
2. Bubble Sort without flag
3. Merge Sort
4. Quick Sort
Enter your choice: 2
Sorted array using Bubble Sort without flag: 68740 8883 6520 62531 87425 89408 60845 86539 80207 54817 66425 65931 59990
 25169 94346 94104 31641 88588 26838 16288 37989 33532 13989 1851 32586 46349 50578 81619 72647 63411 44690 44496 75180
33752 7168 36587 99391 90409 81459 53628 90437 90097 75499 65940 74502 72777 44872 17634 4723 76818

--------------------------------
Process exited after 2.755 seconds with return value 0
Press any key to continue . . .
```



```
68740  8883  6520  62531  87425  89408  60845  86539  80207  54817  66425  65931  59990  25169  94346  94104  31641  885
88  26838  16288  37989  33532  13989  1851  32586  46349  50578  81619  72647  63411  44690  44496  75180  33752  7168
 36587  99391  90409  81459  53628  90437  90097  75499  65940  74502  72777  44872  17634  4723  76818

99996  99996  99996  99996  99996  99996  99996  99996  99997  99997  99997  99997  99997  99998  99998  99998  99998  9
9998  99998  99998  99998  99998  99998  99998  99998  99999  99999  99999  99999  99999  99999  99999  99999  99999  99
999  100000  100000  100000  100000  100000  100000  100000  100000  100000  100000  100000  100000  100000  100000  100
000
--------------------------------
Process exited after 7.633 seconds with return value 0
Press any key to continue . . .
```