Daffodil International University

Lab Reports

Submit to:

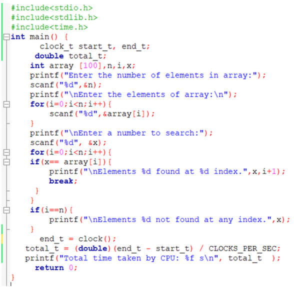
Lecturer name : Ishrat Sultana.
Designation: Lecturer, Department Of Software Engineering
Daffodil International university.

Submit by :

Name : Abu Shalak Ruhan Semester: Spring 2023
ID :- 221-35-835 Batch: 37 E
Course code :SE215 Section : E
Course name : Algorithm Analysis and Design Lab

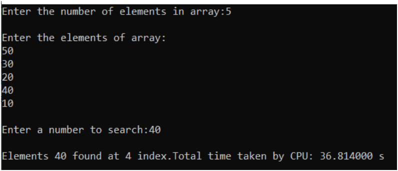
Submission Date: 09-05-2023

1. **Linear Search:**



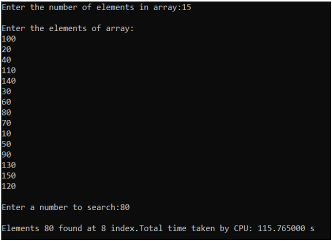
```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int main() {
      clock_t start_t, end_t;
     double total_t;
    int array [100],n,i,x;
    printf("Enter the number of elements in array:");
    scanf("%d",&n);
    printf("\nEnter the elements of array:\n");
    for(i=0;i<n;i++){
        scanf("%d",&array[i]);
    }
    printf("\nEnter a number to search:");
    scanf("%d", &x);
    for(i=0;i<n;i++){
    if(x== array[i]){
        printf("\nElements %d found at %d index.",x,i+1);
        break;
     }
    }
    if(i==n){
        printf("\nElements %d not found at any index.",x);
    }
      end_t = clock();
   total_t = (double)(end_t - start_t) / CLOCKS_PER_SEC;
   printf("Total time taken by CPU: %f s\n", total_t  );
     return 0;
}
```

5 input output:



```
Enter the number of elements in array:5

Enter the elements of array:
50
30
20
40
10

Enter a number to search:40

Elements 40 found at 4 index.Total time taken by CPU: 36.814000 s
```

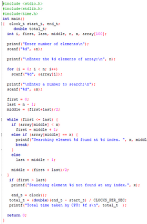
15 Input Output :



```
Enter the number of elements in array:15

Enter the elements of array:
100
20
40
110
140
30
60
80
70
10
50
90
130
150
120

Enter a number to search:80

Elements 80 found at 8 index.Total time taken by CPU: 115.765000 s
```

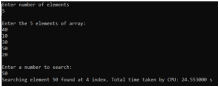
5 input Run time : 36.81 s
15 input Run time : 115.76 s

Binary search :



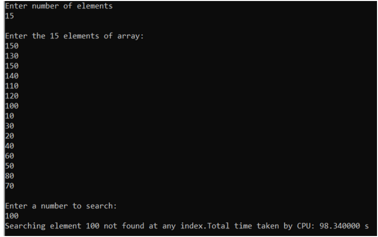
```c
#include <stdio.h>
#include<stdlib.h>
#include<time.h>
int main()
{  clock_t start_t, end_t;
     double total_t;
  int i, first, last, middle, n, x, array[100];

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("\nEnter the %d elements of array:\n", n);

  for (i = 0; i < n; i++)
    scanf("%d", &array[i]);

  printf("\nEnter a number to search:\n");
  scanf("%d", &x);

  first = 0;
  last = n - 1;
  middle = (first+last)/2;

  while (first <= last) {
    if (array[middle] < x)
      first = middle + 1;
    else if (array[middle] == x) {
      printf("Searching element %d found at %d index. ", x, middle+1);
      break;
    }
    else
      last = middle - 1;

    middle = (first + last)/2;
  }
   if (first > last)
    printf("Searching element %d not found at any index.", x);

    end_t = clock();
   total_t = (double)(end_t - start_t) / CLOCKS_PER_SEC;
   printf("Total time taken by CPU: %f s\n", total_t  );

  return 0;
}
```

5 input output:



```
Enter number of elements
5

Enter the 5 elements of array:
40
10
30
50
20

Enter a number to search:
50
Searching element 50 found at 4 index. Total time taken by CPU: 24.553000 s
```

15 input output :

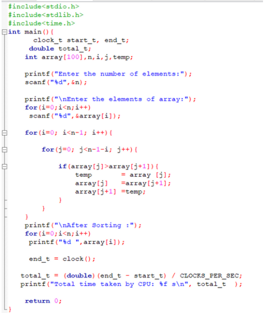


```
Enter number of elements
15

Enter the 15 elements of array:
150
130
150
140
110
120
100
10
30
20
40
60
50
80
70

Enter a number to search:
100
Searching element 100 not found at any index.Total time taken by CPU: 98.340000 s
```

5 input run time ::24.55 s
15 input run time :98.34 s

- Meanwhile i can say for short input linear search is more sufficient than binary search . But for large input binary search is more sufficient.

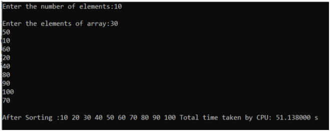
2 . i)Bubble sort :



```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int main(){
      clock_t start_t, end_t;
     double total_t;
    int array[100],n,i,j,temp;

    printf("Enter the number of elements:");
    scanf("%d",&n);

    printf("\nEnter the elements of array:");
    for(i=0;i<n;i++)
     scanf("%d",&array[i]);

    for(i=0; i<n-1; i++){

        for(j=0; j<n-1-i; j++){

            if(array[j]>array[j+1]){
                temp        = array [j];
                array[j]   =array[j+1];
                array[j+1] =temp;
            }
        }
    }
    printf("\nAfter Sorting :");
    for(i=0;i<n;i++)
     printf("%d ",array[i]);

     end_t = clock();

   total_t = (double)(end_t - start_t) / CLOCKS_PER_SEC;
   printf("Total time taken by CPU: %f s\n", total_t  );

    return 0;
}
```

Output :

Time complexity :O(n2)

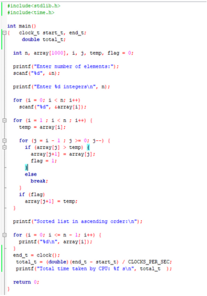


```
Enter the number of elements:10

Enter the elements of array:30
50
10
60
20
40
80
90
100
70

After Sorting :10 20 30 40 50 60 70 80 90 100 Total time taken by CPU: 51.138000 s
```

Run time : 51.13 s

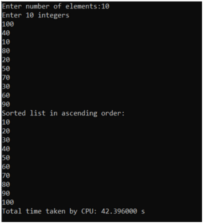
ii) Insertion :

Time complexity :O(n2)



```c
#include<stdlib.h>
#include<time.h>

int main()
{   clock_t start_t, end_t;
     double total_t;

  int n, array[1000], i, j, temp, flag = 0;

  printf("Enter number of elements:");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (i = 0; i < n; i++)
    scanf("%d", &array[i]);

  for (i = 1 ; i < n ; i++) {
    temp = array[i];

    for (j = i - 1 ; j >= 0; j--) {
      if (array[j] > temp) {
        array[j+1] = array[j];
        flag = 1;
      }
      else
        break;
    }
    if (flag)
      array[j+1] = temp;
  }

  printf("Sorted list in ascending order:\n");

  for (i = 0; i <= n - 1; i++) {
   printf("%d\n", array[i]);
  }
  end_t = clock();
   total_t = (double)(end_t - start_t) / CLOCKS_PER_SEC;
   printf("Total time taken by CPU: %f s\n", total_t  );

  return 0;
}
```

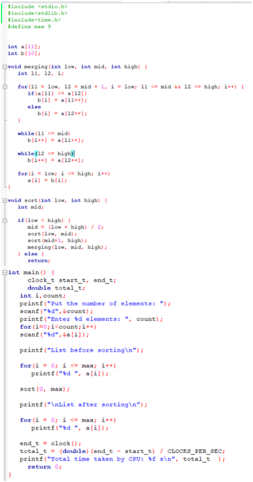
Output:



```
Enter number of elements:10
Enter 10 integers
100
40
10
80
20
50
70
30
60
90
Sorted list in ascending order:
10
20
30
40
50
60
70
80
90
100
Total time taken by CPU: 42.396000 s
```

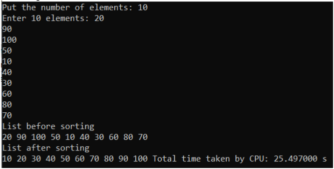
Run time :42.39 s

iii) Merge sort: Time complexity :O(n log n)



```c
#include <stdio.h>
#include<stdlib.h>
#include<time.h>
#define max 9


int a[11];
int b[10];

void merging(int low, int mid, int high) {
   int l1, l2, i;

   for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {
      if(a[l1] <= a[l2])
         b[i] = a[l1++];
      else
         b[i] = a[l2++];
   }

   while(l1 <= mid)
      b[i++] = a[l1++];

   while(l2 <= high)
      b[i++] = a[l2++];

   for(i = low; i <= high; i++)
      a[i] = b[i];
}

void sort(int low, int high) {
   int mid;

   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
int main() {
      clock_t start_t, end_t;
      double total_t;
    int i,count;
    printf("Put the number of elements: ");
    scanf("%d",&count);
    printf("Enter %d elements: ", count);
    for(i=0;i<count;i++)
    scanf("%d",&a[i]);

    printf("List before sorting\n");

    for(i = 0; i <= max; i++)
       printf("%d ", a[i]);

    sort(0, max);

    printf("\nList after sorting\n");

    for(i = 0; i <= max; i++)
       printf("%d ", a[i]);

    end_t = clock();
    total_t = (double)(end_t - start_t) / CLOCKS_PER_SEC;
    printf("Total time taken by CPU: %f s\n", total_t  );
      return 0;
}
```

Output:

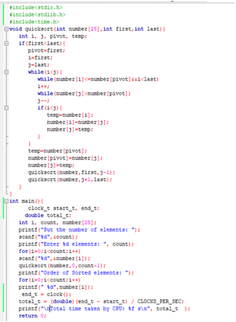


```
Put the number of elements: 10
Enter 10 elements: 20
90
100
50
10
40
30
60
80
70
List before sorting
20 90 100 50 10 40 30 60 80 70
List after sorting
10 20 30 40 50 60 70 80 90 100 Total time taken by CPU: 25.497000 s
```

Run time : 25.49 s

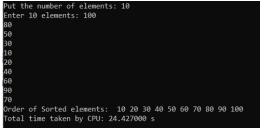
iv)Quick sort:

Time complexity : O(n2)



```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void quicksort(int number[25],int first,int last){
    int i, j, pivot, temp;
    if(first<last){
        pivot=first;
        i=first;
        j=last;
        while(i<j){
            while(number[i]<=number[pivot]&&i<last)
            i++;
            while(number[j]>number[pivot])
            j--;
            if(i<j){
                temp=number[i];
                number[i]=number[j];
                number[j]=temp;
            }
        }
        temp=number[pivot];
        number[pivot]=number[j];
        number[j]=temp;
        quicksort(number,first,j-1);
        quicksort(number,j+1,last);
    }
}
int main(){
        clock_t start_t, end_t;
      double total_t;
    int i, count, number[25];
    printf("Put the number of elements: ");
    scanf("%d",&count);
    printf("Enter %d elements: ", count);
    for(i=0;i<count;i++)
    scanf("%d",&number[i]);
    quicksort(number,0,count-1);
    printf("Order of Sorted elements: ");
    for(i=0;i<count;i++)
    printf(" %d",number[i]);
     end_t = clock();
    total_t = (double)(end_t - start_t) / CLOCKS_PER_SEC;
    printf("\nTotal time taken by CPU: %f s\n", total_t  );
    return 0;
```

Output:



```
Put the number of elements: 10
Enter 10 elements: 100
80
50
30
10
20
40
60
90
70
Order of Sorted elements:  10 20 30 40 50 60 70 80 90 100
Total time taken by CPU: 24.427000 s
```

Run time : 24.42 s

Short discussion:

> Bubble sort is an inefficient sorting algorithm that works by repeatedly swapping adjacent elements if the are in the wrong order until the entire list is sorted.

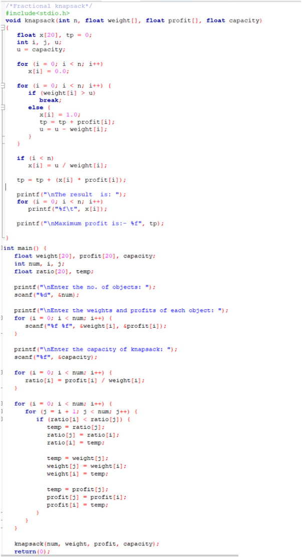
> Insertion sort works by sorting an array by iteratively inserting each element into its proper position in a sorted subarray to its left.

> Merge sort is a divide and conquer algorithm that works by recursively dividing the input list into smaller sublists , sorting them ,and then merging them back together .

>quicksort is also divide and conquer algorithm that works by selecting a pivot element from the list and partitioning the list into two sublists - one containing elements smaller than the pivot and the other containing elements larger than the pivot.

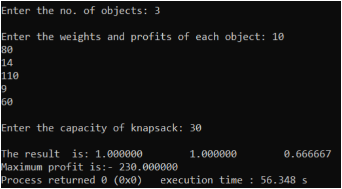
In summary, both bubble sort and insertion sort have a time complexity of O(n2) and are not very efficient for large datasets. Merge sort and quicksort have a time complexity of O(n log n), making them more efficient for large datasets .However ,quicksort has the potential to degrade to O(n2) int worst case scenario ,while merge sort always maintain a time com plexity of 0(n log n).

3 Fractional Knapsack:



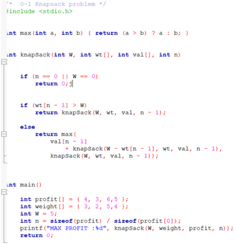
```c
/*Fractional knapsack*/
#include<stdio.h>
void knapsack(int n, float weight[], float profit[], float capacity)
{
   float x[20], tp = 0;
   int i, j, u;
   u = capacity;

   for (i = 0; i < n; i++)
      x[i] = 0.0;

   for (i = 0; i < n; i++) {
      if (weight[i] > u)
         break;
      else {
         x[i] = 1.0;
         tp = tp + profit[i];
         u = u - weight[i];
      }
   }

   if (i < n)
      x[i] = u / weight[i];

   tp = tp + (x[i] * profit[i]);

   printf("\nThe result  is: ");
   for (i = 0; i < n; i++)
      printf("%f\t", x[i]);

   printf("\nMaximum profit is:- %f", tp);

}
int main() {
   float weight[20], profit[20], capacity;
   int num, i, j;
   float ratio[20], temp;

   printf("\nEnter the no. of objects: ");
   scanf("%d", &num);

   printf("\nEnter the weights and profits of each object: ");
   for (i = 0; i < num; i++) {
      scanf("%f %f", &weight[i], &profit[i]);
   }

   printf("\nEnter the capacity of knapsack: ");
   scanf("%f", &capacity);

   for (i = 0; i < num; i++) {
      ratio[i] = profit[i] / weight[i];
   }

   for (i = 0; i < num; i++) {
      for (j = i + 1; j < num; j++) {
         if (ratio[i] < ratio[j]) {
            temp = ratio[j];
            ratio[j] = ratio[i];
            ratio[i] = temp;

            temp = weight[j];
            weight[j] = weight[i];
            weight[i] = temp;

            temp = profit[j];
            profit[j] = profit[i];
            profit[i] = temp;
         }
      }
   }

   knapsack(num, weight, profit, capacity);
   return(0);
```

Output:



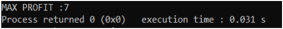
```
Enter the no. of objects: 3

Enter the weights and profits of each object: 10
80
14
110
9
60

Enter the capacity of knapsack: 30

The result  is: 1.000000        1.000000        0.666667
Maximum profit is:- 230.000000
Process returned 0 (0x0)   execution time : 56.348 s
```

4. 0/1 knapsack :



```c
/*  0-1 Knapsack problem */
#include <stdio.h>


int max(int a, int b) { return (a > b) ? a : b; }


int knapSack(int W, int wt[], int val[], int n)


    if (n == 0 || W == 0)
        return 0;j


    if (wt[n - 1] > W)
        return knapSack(W, wt, val, n - 1);

    else
        return max(
            val[n - 1]
                + knapSack(W - wt[n - 1], wt, val, n - 1),
            knapSack(W, wt, val, n - 1));



int main()

    int profit[] = { 4, 3, 6,5 };
    int weight[] = { 3, 2, 5,4 };
    int W = 5;
    int n = sizeof(profit) / sizeof(profit[0]);
    printf("MAX PROFIT :%d", knapSack(W, weight, profit, n));
    return 0;
```

Output:



```
MAX PROFIT :7
Process returned 0 (0x0)   execution time : 0.031 s
```

The end