# Analysis of Algorithm

**Insertion Sort**

Note: analysis to end part is not right giving unexpected results rest of the program is right

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

void main() {
    int array[100];
    int
i,n,item,j,moves=0,comparisons=0,max_comp,max_moves,avg_percent,comp_percent,move_perc
ent;

    max_comp = n*(n-1);
    max_moves = n-1;
    //get array size from user
    printf("Enter size of array: ");
    scanf("%d",&n);

    // store elements in array using for loop
    for(i=0;i<n;i++) {
    printf("Element at location(%d): ",i+1);
    scanf("%d",&array[i]);
    }

    // insertion sort algorithm
    for (j=1;j<n;j++) {
        item = array[j];
        i=j-1;
        while ((i>=0) && (item<array[i])) {
            array[i+1] = array[i];
            i--;
            comparisons++;
        }
        array[i+1]=item;
        moves++;
    }

    //displaying sorted array
    printf("\n--------Sorted Array-------\n");
    for(i=0;i<n;i++) {
        printf("%d\n",array[i]);
    }

    //analysis part
    printf("\n--------Analysis-------\n");
    printf("Comparisons: %d\n",comparisons);
    printf("Moves: %d\n",moves);

    //Complexity
    comp_percent = (comparisons*100)/max_comp;
    move_percent = (moves*100)/max_moves;

    avg_percent = (comp_percent+move_percent)/2;

    printf("\n--------Case-------\n");
    switch(avg_percent) {
        case 0 ... 33:
        printf("Best Case");
        break;

        case 34 ... 66:
        printf("Average Case");
        break;

        case 67 ... 100:
        printf("Worst Case");
        break;

        default:
        printf("Invalid input");
        break;
    }
}
```

**Max Min Algorithm**

```
#include<stdio.h>

void MaxMin(int,int); //function prototype

int array[100];
int max,min;

void main() {
int num,i;
printf("Enter the size of array: ");
scanf("%d",&num);
for(i=1;i<=num;i++) {
  printf("Enter element at location %d: ",i);
  scanf("%d",&array[i]);
}
max = array[0];
min = array[0];
MaxMin(1,num);
printf("\nLargest element in array: %d\n",max);
printf("Smallest element in array: %d\n",min);
}

void MaxMin(int start,int end) { //start and ends are indices of array
  int max1,min1,mid;

  if(start==end) {
    max1 = min1 = start;
  }
  else {
    if(start==end-1) {
      if(array[start]>array[end]) {
        max = array[start];
        min = array[end];
      }
      else {
        max = array[end];
        min = array[start];
      }
    }
    else {
      mid=(start+end)/2;
      MaxMin(start,mid);
      max1 = max;
      min1 = min;
      MaxMin(mid+1,end);
      if(max<max1) {
        max = max1;
      }
      if(min>min1) {
        min = min1;
      }
    }
  }
}
```

**Fractional Knapsack**

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

struct knapsack {
    char id;
    int profit;
    int weight;
    float ratio;
};
struct knapsack items[10],temp;
void main() {
    int n,i,j,capacity,weight;
    float maxprofit=0;
    printf("Enter capacity of knapsack: ");
    scanf("%d",&capacity);
    printf("Enter number of Items: ");
    scanf("%d",&n);

    // getting profit and weight data from user and calculting pi/wi ratio
    for(i=0;i<n;i++) {
    printf("Enter Profit and Weight of Item %c: ",i+65);
    scanf("%d %d",&items[i].profit,&items[i].weight);
    items[i].id = i+65;
```

```c
    items[i].ratio = (float) items[i].profit/items[i].weight;
    }

    // printing data entered by uset
    printf("\n---------Entered Data------------\n");
    printf("Items\tProfit\tWeight\tPi/wi\n");
    for(i=0;i<n;i++) {
    printf("%c\t%d\t%d\t%0.3f\n",items[i].id,items[i].profit,items[i].weight,items[i]
.ratio);
    }

    // sorting the table according to the pi/wi ratio in descending order using
bubble sort
    for(i=0;i<n;i++) {
        for(j=0;j<n-i-1;j++) {
            if(items[j].ratio<items[j+1].ratio) {
                temp = items[j];
                items[j] = items[j+1];
                items[j+1] = temp;
            }
        }
    }

    // printing the data after sorting
    printf("\n---------Sorted Data------------\n");
    printf("Items\tProfit\tWeight\tPi/wi\n");
    for(i=0;i<n;i++) {
    printf("%c\t%d\t%d\t%0.3f\n",items[i].id,items[i].profit,items[i].weight,items[i]
.ratio);
    }

    // actual knapsack algorithm
    int currentWeight =0;
    int currentSpace;
    for(i=0;i<n-1;i++) {
        if((currentWeight+items[i].weight)<= capacity) {
            currentWeight+=items[i].weight;
            maxprofit+=items[i].profit;
        }
        else {
            currentSpace = capacity- currentWeight;
            maxprofit += (float) (items[i].profit*currentSpace)/items[i].weight;
        }
    }
    printf("\nMax Profit: %0.3f\n",maxprofit);
}
```

# Operating System

**First Come First Serve (Practical 4)**

```c
#include <stdio.h>
void main()
{
    int p[20],bt[20],tat[20],total=0,wt[20],i,j,n,temp;
    float avg_wt,avg_tat;

    printf("Enter the number of processes: ");
    scanf("%d",&n);
    for(i=0;i<n;i++) {
        printf("Enter Burst Time of Process P%d: ",i+1);
        scanf("%d",&bt[i]);
        p[i] = i+1;
    }
    // waiting time
    wt[0]=0;
    for(i=1;i<n;i++) {
        wt[i] = 0;
        for(j=0;j<i;j++) {
            wt[i]+=bt[j];
        }
        total+=wt[i];
    }
    avg_wt = (float) total/n;
    total= 0;
    printf("\n-----------TABLE------------\n");
    printf("P\tBT\tWT\tTAT\n");
    for(i=0;i<n;i++) {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("P%d\t%d\t%d\t%d\n",p[i],bt[i],wt[i],tat[i]);
    }
    avg_tat = (float) total/n;
    printf("\nAverage Waiting Time: %0.2f ms\n",avg_wt);
    printf("Average Turnaround Time: %0.2f ms\n",avg_tat);
}
```

**Shortest Job First (Practical 5)**

```c
#include <stdio.h>
void main()
{
    int p[20],bt[20],tat[20],total=0,wt[20],i,j,n,temp,pos;
    float avg_wt,avg_tat;

    printf("Enter the number of processes: ");
    scanf("%d",&n);
    for(i=0;i<n;i++) {
        printf("Enter Burst Time of Process P%d: ",i+1);
        scanf("%d",&bt[i]);
        p[i] = i+1;
    }

    // sorting burst times
    for(i=0;i<n;i++){
        for(j=0;j<n-i-1;j++){
            if(bt[j]>bt[j+1]) {
                temp = bt[j];
                bt[j] = bt[j+1];
                bt[j+1]=temp;

                temp = p[j];
                p[j] = p[j+1];
                p[j+1]=temp;
            }
        }
    }
    // waiting time
    wt[0]=0;
    for(i=1;i<n;i++) {
        wt[i] = 0;
        for(j=0;j<i;j++) {
            wt[i]+=bt[j];
        }
        total+=wt[i];
    }
    avg_wt = (float) total/n;
    total= 0;
    printf("\n-----------TABLE------------\n");
    printf("P\tBT\tWT\tTAT\n");
    for(i=0;i<n;i++) {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("P%d\t%d\t%d\t%d\n",p[i],bt[i],wt[i],tat[i]);
    }
    avg_tat = (float) total/n;
    printf("\nAverage Waiting Time: %0.2f ms\n",avg_wt);
    printf("Average Turnaround Time: %0.2f ms\n",avg_tat);
}
```