1. Implement Euclid's, Consecutive integer checking and Modified Euclid's algorithms to find GCD of two nonnegative integers and perform comparative analysis by generating best case and worst case data.

```
#include<stdio.h>
#include<stdlib.h>
#define n1 10
#define n2 100

int c1,c2,c3;



int cic(int m,int n,int sm){
  c1++;
  int x=m%sm;
  int y = n%sm;
  if(x == 0&& y==0)
    return sm;
  else{
    sm--;
    cic(m,n,sm);}
}

int euclid(int m,int n){
  int r;
  c2 =0;
  while(n != 0){
    c2++;
    r = m%n;
    m = n;
    n = r;
  }
  return m;
}

int repsub(int m,int n){
c3 =0;
  while(m!=n){
    c3++;
    if(m > n)
      m = m-n;
    else
      n = n-m;
  }
  return m;
}

void analysis(){
  FILE *f1,*f2,*f3,*f4,*f5,*f6;
  int max1 =0, max2=0, max3 =0,min1 = 100000,min2 =100000,min3 =100000;
  int m,n,sm,x;
  f1 = fopen("BC1.txt","a");
  f2 = fopen("WC1.txt","a");
  f3 = fopen("BC2.txt","a");
  f4 = fopen("WC2.txt","a");
  f5 = fopen("BC3.txt","a");
  f6 = fopen("WC3.txt","a");
  for(x = n1;x<=n2;x+=10){
       max1 = max2= max3 = 0;min1 = min2 =min3 =100000;
    for(int i = 2;i<= x;i++){
      for(int j = 2;j<= x;j++){
        m=i;n=j;
        sm = (m>n)?n:m;
        c3 = 0;c1 =0;c2 =0;
        cic(m,n,sm);
        euclid(m,n);
        repsub(m,n);
        max1 = c1 > max1?c1:max1;
        min1 = c1 < min1?c1:min1;
        max2 = c2 > max2?c2:max2;
        min2 = c2 < min2?c2:min2;
        max3 = c3 > max3?c3:max3;
        min3 = c3 < min3?c3:min3;
      }
    }
    fprintf(f1,"%d\t%d\n",x,min1);
```

```
    fprintf(f2,"%d\t%d\n",x,max1);
    fprintf(f3,"%d\t%d\n",x,min2);
    fprintf(f4,"%d\t%d\n",x,max2);
    fprintf(f5,"%d\t%d\n",x,min3);
    fprintf(f6,"%d\t%d\n",x,max3);
  }system("gnuplot>load 'command1.txt'");
  fclose(f1);
  fclose(f2);
  fclose(f3);
  fclose(f4);
  fclose(f5);
  fclose(f6);
}

void correctness(){
    int m,n;
    printf("enter two numbers: ");
    scanf("%d %d",&m,&n);
    int sm = m>n?n:m;
    int res=cic(m,n,sm);
    printf("cosecutive integer checking = %d\n",res);
    res = euclid(m,n);
    printf("Euclid's = %d\n",res);
    res = repsub(m,n);
    printf("Repetitive subtraction = %d\n",res);
}

void main(){
    int ch;
    printf("1.Analysis\t\t2.correctness\t\t0.Exit\n");
    for(;;){
    printf("enter choice: ");
    scanf("%d",&ch);
    switch(ch){
        case 1:analysis();break;
        case 2:correctness();break;
        case 0:printf("exiting\n");exit(0);
        default:printf("invalid choice.\n");break;
        }
    }
}
```

2. Implement the following searching algorithms and perform their analysis by generating best case and worst case data. a) Sequential Search b) Binary Search(Recursive)

```
//a
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define n1 10
#define n2 100

int cnt;

int search(int *a,int n,int key){
    for(int i=0;i<n;i++){
        cnt++;
        if(key==a[i])
            return i+1;
    }
    return -1;
}

void analysis(){
    FILE *f1,*f2;
    int *a,n,key;
    f1=fopen("BC.txt","a");
    f2=fopen("WC.txt","a");
    for(n=n1;n<=n2;n+=10){
        a=(int*)malloc(n*sizeof(int));

        for(int i=0;i<n;i++)
            a[i]=rand()%100;
            //BEST CASE
            key=a[0];
            cnt = 0;
            search(a,n,key);
            fprintf(f1,"%d\t%d\n",n,cnt);
            //WORSTCASE
```

```
            key=999;
            cnt = 0;
            search(a,n,key);
            fprintf(f2,"%d\t%d\n",n,cnt);
    }system("gnuplot>load 'command.txt'");
    fclose(f1);
    fclose(f2);
}

void correctness(){
  int a[20],n,key,pos;
  printf("enter the number of elements required: ");
  scanf("%d",&n);
  printf("enter the elements: ");
  for(int i=0;i<n;i++)
    scanf("%d",&a[i]);
  printf("enter the key to search: ");
  scanf("%d",&key);
  pos = search(a,n,key);
  pos > 0?printf("key found at position %d\n",pos):printf("not found!!\n");
}

void main(){
  int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for(;;){
    printf("enter choice: ");
    scanf("%d",&ch);
    switch(ch){
      case 1:analysis();break;
      case 2:correctness();break;
      case 0:printf("exiting..\n");exit(0);
      default:printf("wrong choice!!\n");break;
    }
  }
}

//b
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define n1 10
#define n2 100

int cnt;

int search(int f,int l,int *a,int key){
    cnt++;
    if(f>l)
    return -1;
    int m=(f+l)/2;
    if(a[m]==key)
    return m+1;
    if(a[m]<key)
    return search(m+1,l,a,key);
    else
    return search(f,m-1,a,key);
}

void analysis(){
    int key,n;
    FILE *f1,*f2;
    f1=fopen("BC.txt","a");
    f2=fopen("WC.txt","a");
    for(n=n1;n<=n2;n+=10){
        int *a=(int*)malloc(n*sizeof(int));

        for(int i =0;i<n;i++)
          a[i] = i+1;
        //BEST CASE
        key=a[(n-1)/2];
        cnt=0;
        search(0,n-1,a,key);
        fprintf(f1,"%d\t%d\n",n,cnt);
        //WORST CASE
        key=999;
        cnt=0;
        search(0,n-1,a,key);
```

```
        fprintf(f2,"%d\t%d\n",n,cnt);
    }//system("gnuplot>load 'command.txt'");
    fclose(f1);
    fclose(f2);
}


void correctness(){
  int a[20],n,key,pos;
  printf("enter the number of elements required: ");
  scanf("%d",&n);
  printf("enter the elements in ascending order: ");
  for(int i=0;i<n;i++)
    scanf("%d",&a[i]);
  printf("enter the key to search: ");
  scanf("%d",&key);
  pos = search(0,n-1,a,key);
  pos > 0?printf("key found at position %d\n",pos):printf("not found!!\n");
}

void main(){
  int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for(;;){
    printf("enter choice: ");
    scanf("%d",&ch);
    switch(ch){
      case 1:analysis();break;
      case 2:correctness();break;
      case 0:printf("exiting..\n");exit(0);
      default:printf("wrong choice!!\n");break;
    }
  }
}
```

3. Implement the following elementary sorting algorithms and perform their analysis by generating best case and worst case data. (Note: Any two may be asked in the test/exam)
a) Selection Sort b) Bubble Sort c) Insertion Sort

```
//a
#include<stdio.h>
#include<stdlib.h>
#define n1 10
#define n2 100

int cnt;

void sort(int *a,int n){
    int p,t;
    for(int i=0;i<n-1;i++){
        p=i;
        for(int j=i+1;j<n;j++){
            cnt++;
            if(a[j]<a[p]){
                p=j;
            }}
            t=a[p];
            a[p]=a[i];
            a[i]=t;
        }
}

void analysis(){
    int *a,n;
    FILE *f1,*f2;
    f1=fopen("BC.txt","a");
    f2=fopen("WC.txt","a");
    for(n=n1;n<=n2;n+=10){
        a=(int*)malloc(n*sizeof(int));
        //BEST CASE
        for(int i=0;i<n;i++)
          a[i] = i+1;
        cnt = 0;
        sort(a,n);
        fprintf(f1,"%d\t%d\n",n,cnt);
        //WORST CASE
        for(int i=n-1;i>=0;i--)
          a[i] = n-i+1;
        cnt = 0;
```

```
        sort(a,n);
        fprintf(f2,"%d\t%d\n",n,cnt);
    }system("gnuplot>load 'command.txt'");
    fclose(f1);
    fclose(f2);
}

void correctness(){
  int a[20],n,key,pos;
  printf("enter the number of elements required: ");
  scanf("%d",&n);
  printf("enter the elements: ");
  for(int i=0;i<n;i++)
    scanf("%d",&a[i]);
  printf("array elements after sorting:\n");
  sort(a,n);
  for(int i=0;i<n;i++)
    printf("%d\t",a[i]);
  printf("\n");
}

void main(){
  int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for(;;){
    printf("enter choice: ");
    scanf("%d",&ch);
    switch(ch){
      case 1:analysis();break;
      case 2:correctness();break;
      case 0:printf("exiting..\n");exit(0);
      default:printf("wrong choice!!\n");break;
    }
  }
}

// b
#include <stdio.h>
#include <stdlib.h>
#define n1 10
#define n2 100

int cnt;

void sort(int *a, int n) {
  int t, s = 0;
  for (int i = 0; i < n - 1; i++) {
    s = 0;
    for (int j = 0; j < n - i - 1; j++) {
      cnt++;
      if (a[j] > a[j + 1]) {
        t = a[j];
        a[j] = a[j + 1];
        a[j + 1] = t;
        s = 1;
      }
    }
    if (s == 0)
      break;
  }
}

void analysis() {
  int *a, n;
  FILE *f1, *f2;
  f1 = fopen("BC.txt", "a");
  f2 = fopen("WC.txt", "a");
  for (n = n1; n <= n2; n += 10) {
    a = (int *)malloc(n * sizeof(int));
    // BEST CASE
    for (int i = 0; i < n; i++)
      a[i] = i + 1;
    cnt = 0;
    sort(a, n);
    fprintf(f1, "%d\t%d\n", n, cnt);
    // WORST CASE
    for (int i = n - 1; i >= 0; i--)
      a[i] = n - i + 1;
```

```
    cnt = 0;
    sort(a, n);
    fprintf(f2, "%d\t%d\n", n, cnt);
  }
  system("gnuplot>load 'command.txt'");
  fclose(f1);
  fclose(f2);
}

void correctness() {
  int a[20], n, key, pos;
  printf("enter the number of elements required: ");
  scanf("%d", &n);
  printf("enter the elements: ");
  for (int i = 0; i < n; i++)
    scanf("%d", &a[i]);
  printf("array elements after sorting:\n");
  sort(a, n);
  for (int i = 0; i < n; i++)
    printf("%d\t", a[i]);
  printf("\n");
}

void main() {
  int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for (;;) {
    printf("enter choice: ");
    scanf("%d", &ch);
    switch (ch) {
    case 1:
      analysis();
      break;
    case 2:
      correctness();
      break;
    case 0:
      printf("exiting..\n");
      exit(0);
    default:
      printf("wrong choice!!\n");
      break;
    }
  }
}

// c
#include <stdio.h>
#include <stdlib.h>
#define n1 10
#define n2 100

int cnt;

int sort(int *a, int n) {
  int key, j;
  for (int i = 0; i < n; i++) {
    key = a[i];
    for (j = i - 1; j >= 0 && a[j] > key; j--) {
      cnt++;
      a[j + 1] = a[j];
    }
    a[j + 1] = key;
  }
}

void analysis() {
  int *a, n;
  FILE *f1, *f2;
  f1 = fopen("BC.txt", "a");
  f2 = fopen("WC.txt", "a");
  for (n = n1; n <= n2; n += 10) {
    a = (int *)malloc(n * sizeof(int));
    // BEST CASE
    for (int i = 0; i < n; i++)
      a[i] = i + 1;
    cnt = 0;
    sort(a, n);
```

```
    fprintf(f1, "%d\t%d\n", n, cnt);
    // WORST CASE
    for (int i = n - 1; i >= 0; i--)
      a[i] = n - i + 1;
    cnt = 0;
    sort(a, n);
    fprintf(f2, "%d\t%d\n", n, cnt);
  }
  system("gnuplot>load 'command.txt'");
  fclose(f1);
  fclose(f2);
}

void correctness() {
  int a[20], n, key, pos;
  printf("enter the number of elements required: ");
  scanf("%d", &n);
  printf("enter the elements: ");
  for (int i = 0; i < n; i++)
    scanf("%d", &a[i]);
  printf("array elements after sorting:\n");
  sort(a, n);
  for (int i = 0; i < n; i++)
    printf("%d\t", a[i]);
  printf("\n");
}

void main() {
  int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for (;;) {
    printf("enter choice: ");
    scanf("%d", &ch);
    switch (ch) {
    case 1:
      analysis();
      break;
    case 2:
      correctness();
      break;
    case 0:
      printf("exiting..\n");
      exit(0);
    default:
      printf("wrong choice!!\n");
      break;
    }
  }
}
```

4. Implement Brute force string matching algorithm to search for a pattern of length 'M' in a text of length 'N' (M<=N) and perform its analysis by generating best case and worst case data.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define n1 10
#define n2 100

int cnt;

int stringmatch(char *text, char *pat) {
  int n = strlen(text);
  int m = strlen(pat);
  int i, j;
  for (i = 0; i <= n - m; i++) {
    for (j = 0; j < m; j++) {
      cnt++;
      if (text[i + j] != pat[j])
        break;
    }
    if (j == m)
      return i+1;
  }
  return -1;
}

void analysis() {
```

```
  FILE *f1, *f2;
  int n;
  f1 = fopen("BC.txt", "a");
  f2 = fopen("WC.txt", "a");
  for (n = n1; n <= n2; n += 10) {
    char *t = (char *)malloc(101 * sizeof(char));
    char *p = (char *)malloc(n * sizeof(char));
    for (int i = 0; i < 100; i++)
      t[i] = 'a';
    t[100] = '\0';
    // BEST CASE
    for (int i = 0; i < n; i++)
      p[i] = 'a';
    p[n] = '\0';
    cnt = 0;
    stringmatch(t, p);
    fprintf(f1, "%d\t%d\n", n, cnt);
    // WORST CASE
    p[n - 1] = 'b';
    cnt = 0;
    stringmatch(t, p);
    fprintf(f2, "%d\t%d\n", n, cnt);
  } // system("gnuplot>load 'command.txt'");
  fclose(f1);
  fclose(f2);
}

void correctness() {
  char text[50], pat[10];
  printf("enter the text: ");
  scanf("%s", text);
  printf("enter the pattern: ");
  scanf("%s", pat);
  int pos;
  pos = stringmatch(text, pat);
  pos > 0 ? printf("pattern found at position %d of text.\n", pos)
          : printf("pattern not found\n");
}

void main() {
  int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for (;;) {
    printf("enter choice: ");
    scanf("%d", &ch);
    switch (ch) {
    case 1:
      analysis();
      break;
    case 2:
      correctness();
      break;
    case 0:
      printf("exiting..\n");
      exit(0);
    default:
      printf("wrong choice!!\n");
      break;
    }
  }
}
```

5. Implement Merge Sort algorithm and perform its analysis by generating best case and worst case data.

```
#include <stdio.h>
#include <stdlib.h>
#define n1 4
#define n2 1024
int cnt = 0;

void merge(int *a, int *left, int l, int *right, int r) {
  int i = 0, j = 0, k = 0;

  while (i < l && j < r) {
    if (left[i] <= right[j]) {
      a[k++] = left[i++];
    } else {
      a[k++] = right[j++];
```

```
    }
    cnt++;
  }

  while (i < l) {
    a[k++] = left[i++];
  }

  while (j < r) {
    a[k++] = right[j++];
  }
}

int sort(int *a, int n) {
  if (n < 2) {
    return cnt;
  }
  int mid = n / 2;
  int *left = (int *)malloc(mid * sizeof(int));
  int *right = (int *)malloc((n - mid) * sizeof(int));

  for (int i = 0; i < mid; i++) {
    left[i] = a[i];
  }

  for (int i = mid; i < n; i++) {
    right[i - mid] = a[i];
  }

  sort(left, mid);
  sort(right, n - mid);
  merge(a, left, mid, right, n - mid);
  return cnt;
}

void generateworstcase(int *a, int l, int r) {
  if (l < r) {
    int m = (l + r) / 2;
    int *left = (int *)malloc((m - l + 1) * sizeof(int));
    int *right = (int *)malloc((r - m) * sizeof(int));
    for (int i = 0; i <= m - l; i++)
      left[i] = a[i * 2];

    for (int i = 0; i < r - m; i++)
      right[i] = a[i * 2 + 1];
    generateworstcase(left, l, m);
    generateworstcase(right, m + 1, r);
    int i;
    for (i = 0; i <= m - l; i++)
      a[i] = left[i];

    for (int j = 0; j < r - m; j++)
      a[i + j] = right[j];
  }
}

void analysis() {
  int *a, n;
  FILE *f1, *f2;
  f1 = fopen("BC.txt", "a");
  f2 = fopen("WC.txt", "a");
  for (n = n1; n <= n2; n *= 2) {
    a = (int *)malloc(n * sizeof(int));

    // BEST CASE
    for (int i = 0; i < n; i++)
      a[i] = i + 1;
    cnt = 0;
    cnt = sort(a, n);
    fprintf(f1, "%d\t%d\n", n, cnt);

    // WORST CASE
    generateworstcase(a, 0, n - 1);
    cnt = 0;
    cnt = sort(a, n);
    fprintf(f2, "%d\t%d\n", n, cnt);
  } // system("gnuplot>load 'command.txt'");
  fclose(f1);
```

```
  fclose(f2);
}

void correctness() {
  int a[20], n, key, pos;
  printf("enter the number of elements required: ");
  scanf("%d", &n);
  printf("enter the elements: ");
  for (int i = 0; i < n; i++)
    scanf("%d", &a[i]);
  printf("array elements after sorting:\n");
  sort(a, n);
  for (int i = 0; i < n; i++)
    printf("%d\t", a[i]);
  printf("\n");
}

void main() {
  int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for (;;) {
    printf("enter choice: ");
    scanf("%d", &ch);
    switch (ch) {
    case 1:
      analysis();
      break;
    case 2:
      correctness();
      break;
    case 0:
      printf("exiting..\n");
      exit(0);
    default:
      printf("wrong choice!!\n");
      break;
    }
  }
}
```

6. Implement Quick Sort algorithm and perform its by generating best case and worst case data

```
#include <stdio.h>
#include <stdlib.h>
#define n1 10
#define n2 100
    int cnt = 0;

void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}

int partition(int *a, int low, int high) {
  int pivot = a[high];
  int i = (low - 1);
  for (int j = low; j <= high - 1; j++) {
    cnt++;
    if (a[j] < pivot) {
      i++;
      swap(&a[i], &a[j]);
    }
  }
  swap(&a[i + 1], &a[high]);
  return (i + 1);
}

int sort(int *a, int low, int high) {
  if (low < high) {
    int pi = partition(a, low, high);
    sort(a, low, pi - 1);
    sort(a, pi + 1, high);
  }
  return cnt;
}

void analysis() {
  int *a, n;
```

```
  FILE *f1, *f2;
  f1 = fopen("BC.txt", "a");
  f2 = fopen("WC.txt", "a");
  for (n = n1; n <= n2; n += 10) {
    a = (int *)malloc(n * sizeof(int));
    // BEST CASE
    for (int i = 0; i < n; i++)
      a[i] = rand() % 100;
    swap(&a[n - 1], &a[n / 2]);
    cnt = 0;
    cnt = sort(a, 0, n - 1);
    fprintf(f1, "%d\t%d\n", n, cnt);

    // WORST CASE
    for (int i = n - 1; i >= 0; i--)
      a[i] = n - i + 1;
    cnt = 0;
    cnt = sort(a, 0, n - 1);
    fprintf(f2, "%d\t%d\n", n, cnt);
  } // system("gnuplot>load 'command.txt'");
  fclose(f1);
  fclose(f2);
}

void correctness() {
  int a[20], n, key, pos;
  printf("enter the number of elements required: ");
  scanf("%d", &n);
  printf("enter the elements: ");
  for (int i = 0; i < n; i++)
    scanf("%d", &a[i]);
  printf("array elements after sorting:\n");
  sort(a, 0,n-1);
  for (int i = 0; i < n; i++)
    printf("%d\t", a[i]);
  printf("\n");
}

void main() {
  int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for (;;) {
    printf("enter choice: ");
    scanf("%d", &ch);
    switch (ch) {
    case 1:
      analysis();
      break;
    case 2:
      correctness();
      break;
    case 0:
      printf("exiting..\n");
      exit(0);
    default:
      printf("wrong choice!!\n");
      break;
    }
  }
}
```

7. Implement DFS algorithm to check for connectivity and acyclicity of a graph. If not connected, display the connected components. Perform its analysis by generating best case and worst case data.
Note: while showing correctness, input should be given for both connected/disconnected and cyclic/acyclic graphs.

```
#include <stdio.h>
#include <stdlib.h>
#define n1 3
#define n2 10
int a[100][100], visited[100], n, acyclic, cnt = 0;

void analysis();

void dfs(int v) {
  visited[v] = 1;
  for (int i = 1; i <= n; i++) {
    cnt++;
```

```
    if (a[v][i] && visited[i]) {
      acyclic = 0;
      printf("-->%d-->%d\n", v, i);
    }
    if (a[v][i] && !visited[i]) {
      printf("-->%d-->%d\n", v, i);
      dfs(i);
    }
  }
}

void connected_cyclic(int start) {
  int i;
  for (i = 1; i <= n; i++)
    if (!visited[i])
      break;
  if (i == n + 1)
    printf("The graph is connected\n");
  else
    printf("Graph is not connected\n");

  if (acyclic)
    printf("Graph is acyclic\n");
  else
    printf("Graph is cyclic\n");
}

void analysis() {
  int i, j;
  FILE *f1, *f2;
  f1 = fopen("BC.txt", "a");
  f2 = fopen("WC.txt", "a");
  for (n = n1; n <= n2; n += 1) {
    for (i = 1; i <= n; i++)
      visited[i] = 0;
    // BEST CASE
    printf("BEST CASE : vertices:%d\n", n);
    printf("connected components are:\n");
    for (i = 1; i <= n; i++)
      for (j = 1; j <= n; j++)
        if (i == j - 1)
          a[i][j] = 1;
        else
          a[i][j] = 0;

    for (i = 1; i <= n; i++)
      visited[i] = 0;
    acyclic = 1;
    cnt = 0;
    visited[1]=1;
    dfs(1);
    connected_cyclic(1);
    fprintf(f1, "%d\t%d\n", n, cnt);
    // WORST CASE
    printf("WORST CASE : vertices:%d\n", n);
    printf("connected components are:\n");
    for (i = 1; i <= n; i++)
      for (j = 1; j <= n; j++)
        a[i][j] = 1;
    for (i = 1; i <= n; i++)
      visited[i] = 0;
    acyclic = 1;
    cnt = 0;
    visited[1]=1;
    dfs(1);
    connected_cyclic(1);
    fprintf(f2, "%d\t%d\n", n, cnt);
  } // system("gnuplot>load 'command.txt'");
  fclose(f1);
  fclose(f2);
}

void correctness() {
  int i, j, start;
  printf("Enter the number of vertices: ");
  scanf("%d", &n);
  for (i = 1; i <= n; i++)
    visited[i] = 0;
```

```
  printf("Enter the adjacency matrix\n");
  for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
      scanf("%d", &a[i][j]);
  printf("Enter the start vertex: ");
  scanf("%d", &start);
  visited[start] = 1;
  acyclic = 1;
  dfs(start);
  connected_cyclic(start);
}

void main() {
  int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for (;;) {
    printf("enter choice: ");
    scanf("%d", &ch);
    switch (ch) {
    case 1:
      analysis();
      break;
    case 2:
      correctness();
      break;
    case 0:
      printf("exiting..\n");
      exit(0);
    default:
      printf("wrong choice!!\n");
      break;
    }
  }
}
```

8. Implement BFS algorithm to check for connectivity and acyclicity of a graph. If not connected, display the connected components. Perform its analysis by generating best case and worst case data.
Note: while showing correctness, Input should be given for both connected/disconnected and cyclic/acyclic graphs.

```
#include<stdio.h>
#include<stdlib.h>
#define n1 3
#define n2 10
int a[100][100], visited[100], n, acyclic;
int f = 0, r = -1, q[20], n,cnt=0;

void bfs(int v) {
  int i;
  visited[v]=1;
  for (i = 1; i <= n; i++) {
    cnt++;
    if (a[v][i] && visited[i]) {
      acyclic = 0;
      printf("-->%d-->%d\n", v, i);
    }
    if (a[v][i] && !visited[i]) {
      q[++r] = i;
      visited[i]=1;
      printf("-->%d-->%d\n", v, i);
    }
  }
  if (r >= f) {
    visited[q[f]] = 1;
    bfs(q[f++]);
  }
}


void connected_cyclic(int start){
  int i;
  for( i=1;i<=n;i++)
    if(!visited[i])
      break;
  if(i==n+1)
    printf("The graph is connected\n");
  else
    printf("Graph is not connected\n");
```

```
  if(acyclic)
    printf("Graph is acyclic\n");
  else
    printf("Graph is cyclic\n");
}

void analysis(){
  int i,j;
  FILE *f1,*f2;
  f1 = fopen("BC.txt","a");
  f2 = fopen("WC.txt","a");
  for(n=n1;n<=n2;n+=1){
    for(i=1;i<=n;i++)
    visited[i]=0;
    //BEST CASE
     for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
       if(i==j-1)
         a[i][j]=1;
       else
       a[i][j] =0;

    for(i=1;i<=n;i++)
    visited[i]=0;
    acyclic=1;
     cnt=0;
     printf("BEST CASE : vertices:%d\n",n);
     printf("connected components are:\n");
     bfs(1);
     connected_cyclic(1);
     fprintf(f1,"%d\t%d\n",n,cnt);
    //WORST CASE
    for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
         a[i][j] =1;

    for(i=1;i<=n;i++)
    visited[i]=0;
    acyclic=1;
    cnt=0;
    printf("WORST CASE : vertices:%d\n",n);
    printf("connected components are:\n");
    bfs(1);
    connected_cyclic(1);
    fprintf(f2,"%d\t%d\n",n,cnt);
  }//system("gnuplot>load 'command.txt'");
  fclose(f1);
  fclose(f2);
}

void correctness() {
  int i, j, start;
  printf("Enter the number of vertices: ");
  scanf("%d", &n);
  for (i = 1; i <= n; i++)
    visited[i] = 0;
  printf("Enter the adjacency matrix\n");
  for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
      scanf("%d", &a[i][j]);
  printf("Enter the start vertex: ");
  scanf("%d", &start);
  visited[start] = 1;
  acyclic = 1;
  bfs(start);
  connected_cyclic(start);
}

void main() {
  int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for (;;) {
    printf("enter choice: ");
    scanf("%d", &ch);
    switch (ch) {
    case 1:
      analysis();
```

```
      break;
    case 2:
      correctness();
      break;
    case 0:
      printf("exiting..\n");
      exit(0);
    default:
      printf("wrong choice!!\n");
      break;
    }
  }
}
```

9. Implement DFS based algorithm to list the vertices of a directed graph in Topological ordering. Perform its analysis giving minimum 5 graphs with different number of vertices and edges. (starting with 4 vertices).
Note: while showing correctness, input should be given for with and without solution.

```
#include <stdio.h>
#define n1 4
#define n2 8
int graph[40][40], n, visited[40], stack[40], stop,cnt,acyclic;
void dfs(int );

void dfstopo(){
  int i,count=0;
  for (i = 0; i < n; i++)
        if (!visited[i])
            dfs(i);
  if(!acyclic){
    printf("invalid input\n");
    return ;
  }
       printf("Topologically Sorted Order:\n");
       for(i=n-1;i>=0;i--)
              printf("%d   ",stack[i]);
       printf("\n");
}

void dfs(int v){
       visited[v]=1;
       for(int i=0;i<n;i++){
             cnt++;
            if (graph[v][i] && visited[i])
                acyclic = 0;
             if (graph[v][i] && !visited[i])
                    dfs(i);}
       stack[++stop]=v;
}

void correctness(){
       printf("No. of vertices: ");
       scanf("%d", &n);
       printf("Enter adjacency matrix:\n");
       for(int i=0;i<n;i++)
             for(int j=0;j<n;j++)
                    scanf("%d",&graph[i][j]);
   for (int i = 0; i < n; i++)
      visited[i] = 0;
  stop = -1;
  acyclic = 1;
  dfstopo();
}

void analysis() {
  int i, j;
  FILE *f;
  f = fopen("BC.txt", "a");
  for (n = n1; n <= n2; n += 1) {

    for(i=0;i<n;i++)
     for(j=0;j<n;j++)
       if(i==j-1)
         graph[i][j]=1;
       else
          graph[i][j] =0;
    for (i = 0; i < n; i++)
      visited[i] = 0;
```

```
    cnt = 0;
    stop = -1;
    acyclic=1;
    dfstopo();
    fprintf(f, "%d\t%d\n", n, cnt);
  } // system("gnuplot>load 'command.txt'");
  fclose(f);
}

void main() {
  int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for (;;) {
    printf("enter choice: ");
    scanf("%d", &ch);
    switch (ch) {
    case 1:
      analysis();
      break;
    case 2:
      correctness();
      break;
    case 0:
      printf("exiting..\n");
      exit(0);
    default:
      printf("wrong choice!!\n");
      break;
    }
  }
}
```

10. Implement source removal algorithm to list the vertices of a directed graph in Topological ordering. Perform its analysis giving minimum 5 graphs with different number of vertices and edges. (starting with 4 vertices).
Note: Use efficient method to identify the source vertex. While showing correctness, Input should be given for with and without solution.

```
#include <stdio.h>
#include <stdlib.h>
#define n1 4
#define n2 8
int graph[10][10], visited[10], indegree[10], n, cnt;

void sourcetopo() {
  int i, j, count = 0;
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      indegree[i] += graph[j][i];
  for(i=0;i<n;i++)
    if(!indegree[i])
      acyclic = 1;
  if(!acyclic){
    printf("invalid input\n");
    return ;
  }
  printf("\nTopologically sorted order: \n");
  while (count < n) {
    for (i = 0; i < n; i++) {
      if (!visited[i] && !indegree[i]) {
        printf("%d  ", i+1);
        visited[i] = 1;
        for (j = 0; j < n; j++) {
          cnt++;
          if (graph[i][j]) {
            graph[i][j] = 0;
            indegree[j]--;
          }
        }
        count++;
        break;
      }
    }
  }
}

void correctness() {
  printf("Enter no. of vertices: ");
```

```
  scanf("%d", &n);
  printf("Enter adjacency matrix:\n");
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      scanf("%d", &graph[i][j]);
  for (int i = 0; i < n; i++) {
    visited[i] = 0;
    indegree[i] = 0;
  }
  sourcetopo();
}

void analysis() {
  int i, j;
  FILE *f;
  f = fopen("BC.txt", "a");
  for (n = n1; n <= n2; n += 1) {

    for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
        if (i == j - 1)
          graph[i][j] = 1;
        else
          graph[i][j] = 0;
    for (i = 0; i < n; i++) {
      visited[i] = 0;
      indegree[i] = 0;
    }
    cnt = 0;
    sourcetopo();
    fprintf(f, "%d\t%d\n", n, cnt);
  } //system("gnuplot>load 'command.txt'");
  fclose(f);
}

void main() {
  int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for (;;) {
    printf("\nenter choice: ");
    scanf("%d", &ch);
    switch (ch) {
    case 1:
      analysis();
      break;
    case 2:
      correctness();
      break;
    case 0:
      printf("exiting..\n");
      exit(0);
    default:
      printf("wrong choice!!\n");
      break;
    }
  }
}
```

11. Implement heap sort algorithm with bottom-up heap construction. Perform its analysis by generating best case and worst case data.

```
#include <stdio.h>
#include <stdlib.h>
#define n1 10
#define n2 100

int cnt;

void heapify(int *a, int n, int i) {
  cnt++;
  int largest = i;
  int left = 2 * i + 1;
  int right = 2 * i + 2;
  if (left < n && a[left] > a[largest])
    largest = left;
  if (right < n && a[right] > a[largest])
    largest = right;
  if (largest != i) {
```

```
    int temp = a[i];
    a[i] = a[largest];
    a[largest] = temp;

    heapify(a, n, largest);
  }
}

void sort(int *a, int n) {
  for (int i = n / 2 - 1; i >= 0; i--)
    heapify(a, n, i);
  for (int i = n - 1; i >= 0; i--) {
    int temp = a[0];
    a[0] = a[i];
    a[i] = temp;
    heapify(a, i, 0);
  }
}

void analysis(){
    int *a,n;
    FILE *f1,*f2;
    f1=fopen("BC.txt","a");
    f2=fopen("WC.txt","a");
    for(n=n1;n<=n2;n+=10){
        a=(int*)malloc(n*sizeof(int));
        //BEST CASE
        for(int i=n-1;i>=0;i--)
          a[i] = n-i+1;
        cnt = 0;
        sort(a,n);
        fprintf(f1,"%d\t%d\n",n,cnt);
        //WORST CASE
        for(int i=0;i<n;i++)
          a[i] = i+1;
        cnt = 0;
        sort(a,n);
        fprintf(f2,"%d\t%d\n",n,cnt);
    }//system("gnuplot>load 'command.txt'");
    fclose(f1);
    fclose(f2);
}

void correctness() {
  int a[20], n, key, pos;
  printf("enter the number of elements required: ");
  scanf("%d", &n);
  printf("enter the elements: ");
  for (int i = 0; i < n; i++)
    scanf("%d", &a[i]);
  printf("array elements after sorting:\n");
  sort(a, n);
  for (int i = 0; i < n; i++)
    printf("%d\t", a[i]);
  printf("\n");
}

void main() {
  int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for (;;) {
    printf("enter choice: ");
    scanf("%d", &ch);
    switch (ch) {
    case 1:
      analysis();
      break;
    case 2:
      correctness();
      break;
    case 0:
      printf("exiting..\n");
      exit(0);
    default:
      printf("wrong choice!!\n");
      break;
    }
  }
```

}

12. a) Implement Warshall's Algorithm to find the transitive closure of a directed graph and perform its analysis giving minimum 5 graphs with different number of vertices and edges. (starting with 4 vertices).
b) Implement Floyd's Algorithm to find All-pair shortest paths for a graph and perform its analysis giving minimum 5 graphs with different number of vertices and edges(starting with 4 vertices).

```
// a
#include <stdio.h>
#include<stdlib.h>
#define n1 4
#define n2 8
int graph[40][40], n,cnt;

void warshall() {
  for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        cnt++;
        graph[i][j] = (graph[i][j] | (graph[i][k] & graph[k][j]));
      }
    }
  }
}

void analysis() {
  int i, j;
  FILE *f;
  f = fopen("BC.txt", "a");
  for (n = n1; n <= n2; n += 1) {

    for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
        if (i == j)
          graph[i][j] = 0;
        else
          graph[i][j] = rand()%2;

    cnt = 0;
    warshall();
    fprintf(f, "%d\t%d\n", n, cnt);
  } //system("gnuplot>load 'command.txt'");
  fclose(f);
}

void correctness() {
  printf("No. of vertices: ");
  scanf("%d", &n);
  printf("Enter adjacency matrix:\n");
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      scanf("%d", &graph[i][j]);
  printf("Applying Warshall's Algorithm\n");
  warshall();
  printf("Transitive Closure Matrix:\n");
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      printf("%d ", graph[i][j]);
    }
    printf("\n");
  }
}

void main() {
  int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for (;;) {
    printf("enter choice: ");
    scanf("%d", &ch);
    switch (ch) {
    case 1:
      analysis();
      break;
    case 2:
      correctness();
      break;
```

```
    case 0:
      printf("exiting..\n");
      exit(0);
    default:
      printf("wrong choice!!\n");
      break;
    }
  }
}

//b
#include <stdio.h>
#include<stdlib.h>
#define n1 4
#define n2 8
int graph[40][40], n,cnt;

void floyd() {
  for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        cnt++;
        if (graph[i][k] + graph[k][j] < graph[i][j])
          graph[i][j] = graph[i][k] + graph[k][j];
      }
    }
  }
}

void analysis() {
  int i, j;
  FILE *f;
  f = fopen("BC.txt", "a");
  for (n = n1; n <= n2; n += 1) {

    for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
        if (i == j)
          graph[i][j] = 0;
        else
          graph[i][j] = rand()%99;

    cnt = 0;
    floyd();
    fprintf(f, "%d\t%d\n", n, cnt);
  } //system("gnuplot>load 'command.txt'");
  fclose(f);
}

void correctness() {
  printf("No. of vertices: ");
  scanf("%d", &n);
  printf("Enter adjacency matrix:\n");
  printf("enter 999 for infinity: \n");
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      scanf("%d", &graph[i][j]);

  printf("Applying Floyd's Algorithm\n");
  floyd();
  printf("All Pair Shortest Path Matrix:\n");
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      printf("%d ", graph[i][j]);
    }
    printf("\n");
  }
}

void main() {
  int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for (;;) {
    printf("enter choice: ");
    scanf("%d", &ch);
    switch (ch) {
    case 1:
      analysis();
```

```
      break;
    case 2:
      correctness();
      break;
    case 0:
      printf("exiting..\n");
      exit(0);
    default:
      printf("wrong choice!!\n");
      break;
    }
  }
}
```

13. a) Implement bottom up Dynamic Programming algorithm to solve Knapsack problem and perform its analysis with different instances (different number of items and Capacity, starting with 4 items)
b) Implement a Dynamic Programming algorithm with Memory function to solve Knapsack problem and perform its analysis with different instances (different number of items and Capacity, starting with 4 items).

```
//a
#include <stdio.h>
#include <stdlib.h>
#define n1 4
#define n2 10
int t[100][100], v[100], w[100], n, m,cnt;

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

void knapsack(){
  int i,j;
  for(i=0;i<n+1;i++){
              for(j=0;j<m+1;j++){
                  cnt++;
                     if (i==0||j==0)
                            t[i][j] = 0;
                     else if (j<w[i])
                            t[i][j] = t[i-1][j];
                     else
                            t[i][j] = max(t[i-1][j], v[i]+t[i-1][j-w[i]]);
              }
       }
       printf("table\n");
              for(i=0;i<n+1;i++){
              for(j=0;j<m+1;j++){
                  printf("%d\t",t[i][j]);
                  }
              printf("\n");
              }

       printf("Maximum Value: %d\n",t[n][m]);
}

void correctness(){
  int i,j;
       printf("No. of Items: ");
       scanf("%d",&n);
       printf("Capacity: ");
       scanf("%d",&m);

       printf("Weight\tValue\n");
       for(i=1;i<n+1;i++)
              scanf("%d\t%d",&w[i],&v[i]);
  knapsack();

}

void analysis(){
    int i,j;
    FILE *f;
    f = fopen("BC.txt","a");
    m = 10;
    for(n=n1;n<=n2;n++){
        for(i=1;i<n+1;i++){
            w[i] = rand()%12;
            v[i] = rand()%100;
```

```
            printf("%d\t%d\n",w[i],v[i]);
        }
        cnt = 0;
        knapsack();
        fprintf(f,"%d\t%d\n",n,cnt);
    }
}

void main(){
    int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for (;;) {
    printf("enter choice: ");
    scanf("%d", &ch);
    switch (ch) {
    case 1:
      analysis();
      break;
    case 2:
      correctness();
      break;
    case 0:
      printf("exiting..\n");
      exit(0);
    default:
      printf("wrong choice!!\n");
      break;
    }
  }
}

//b
#include <stdio.h>
#include <stdlib.h>
#define n1 4
#define n2 10
int t[100][100], v[100], w[100], n, m,cnt;

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

int knap(int i, int j){
       if (t[i][j]==-1){
              if (j<w[i])
                     t[i][j] = knap(i-1,j);
              else
                     t[i][j] = max(knap(i-1,j),v[i]+knap(i-1,j-w[i]));
       }
       return t[i][j];
}

void knapsack(){
  int i,j;
  for(i=0;i<n+1;i++){
              for(j=0;j<m+1;j++){
      cnt++;
                     if (i==0||j==0)
                            t[i][j]=0;
                     else
                            t[i][j]=-1;
              }
       }
       printf("Maximum Value: %d\n",knap(n,m));
       printf("table\n");
              for(i=0;i<n+1;i++){
              for(j=0;j<m+1;j++){
                  printf("%d\t",t[i][j]);
              }
              printf("\n");
              }
}

void correctness(){
  int i,j;
       printf("No. of Items: ");
       scanf("%d",&n);
       printf("Capacity: ");
```

```
       scanf("%d",&m);

       printf("Weight\tValue\n");
       for(i=1;i<n+1;i++)
             scanf("%d\t%d",&w[i],&v[i]);
       knapsack();
}

void analysis(){
  int i,j;
    FILE *f;
    f = fopen("BC.txt","a");
    m = 10;
    for(n=n1;n<=n2;n++){
      printf("Weight\tValue\n");
        for(i=1;i<n+1;i++){
            w[i] = rand()%12;
            v[i] = rand()%100;
            printf("%d\t\t%d\n",w[i],v[i]);
        }
        cnt = 0;
        knapsack();
        fprintf(f,"%d\t%d\n",n,cnt);
    }

}

void main(){
    int ch;
  printf("1.analysis\t\t2.correctness\t\t0.exit\n");
  for (;;) {
    printf("enter choice: ");
    scanf("%d", &ch);
    switch (ch) {
    case 1:
      analysis();
      break;
    case 2:
      correctness();
      break;
    case 0:
      printf("exiting..\n");
      exit(0);
    default:
      printf("wrong choice!!\n");
      break;
    }
  }
}
```

14. Implement Prim's algorithm to find Minimum Spanning Tree of a graph and perform its analysis giving minimum 5 graphs with different number of vertices and edges (starting with 4 vertices)

```
#include <stdio.h>
#include<stdlib.h>
#define n1 4
#define n2 8
int cost[40][40], n, visited[40],cnt;

void prims() {
  int i, j, edges = 0;
  int a, b, min, min_cost = 0;
  visited[0] = 1;
  while (edges < n - 1) {
    min = 9999;
    for (i = 0; i < n; i++) {
      if (visited[i]) {
        for (j = 0; j < n; j++) {
          cnt++;
          if (cost[i][j] && min > cost[i][j] && !visited[j]) {
            min = cost[i][j];
            a = i;
            b = j;
          }
        }
      }
    }
    printf("%d-->%d | Cost: %d\n", a, b, min);
```

```
    visited[b] = 1;
    min_cost += min;
    edges++;
  }
  printf("Minimum Cost: %d\n", min_cost);
}

void correctness() {
  printf("No. of vertices: ");
  scanf("%d", &n);
  printf("Enter cost matrix:\n");
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      scanf("%d", &cost[i][j]);
  for(int i=0;i<n;i++)
    visited[i] = 0;
  prims();
}

void analysis() {
    int i, j;
    FILE *f;
    f = fopen("BC.txt", "a");
    for (n = n1; n <= n2; n += 1) {

      for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
          if (i == j)
            cost[i][j] = 0;
          else
            cost[i][j] = rand() % 10;

      for(int i=0;i<n;i++)
        visited[i] = 0;
      cnt = 0;
      prims();
      fprintf(f, "%d\t%d\n", n, cnt);
    } // system("gnuplot>load 'command.txt'");
    fclose(f);
 }

void main() {
    int ch;
    printf("1.analysis\t\t2.correctness\t\t0.exit\n");
    for (;;) {
      printf("enter choice: ");
      scanf("%d", &ch);
      switch (ch) {
      case 1:
        analysis();
        break;
      case 2:
        correctness();
        break;
      case 0:
        printf("exiting..\n");
        exit(0);
      default:
        printf("wrong choice!!\n");
        break;
      }
    }
  }
```

15. Implement Dijkstra's algorithm to find the shortest path from a given source to all other vertices and perform its analysis giving minimum 5 graphs with different number of vertices and edges(starting with 4 vertices).

```
#include <stdio.h>
#include<stdlib.h>
#define n1 4
#define n2 8
int graph[20][20], visited[20], dist[20], n,cnt;

int mindist() {
  int min = 9999, md;
  for (int i = 0; i < n; i++) {
    if (!visited[i] && dist[i] < min) {
      min = dist[i];
```

```
      md = i;
    }
  }
  return md;
}

void dijkstra(int v) {
  for (int i = 0; i < n; i++) {
    dist[i] = 9999;
    visited[i] = 0;
  }
  dist[v] = 0;
  for (int i=0; i < n; i++) {
    int j = mindist();
    visited[j] = 1;
    for (int i = 0; i < n; i++) {
      cnt++;
      if (!visited[i] && graph[j][i] && dist[j] != 9999 &&
          dist[j] + graph[j][i] < dist[i]) {
        dist[i] = dist[j] + graph[j][i];
      }
    }
  }
  printf("Shortest distances from source vertex %d:\n", v);
  for (int i = 0; i < n; i++)
    printf("Vertex %d: %d\n", i, dist[i]);
}

void correctness() {
  int start;
  printf("Enter the number of vertices: ");
  scanf("%d", &n);
  printf("Enter the adjacency matrix :\n");
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      scanf("%d", &graph[i][j]);
    }
  }
  printf("Enter the source vertex: ");
  scanf("%d", &start);
  dijkstra(start);
}

void analysis() {
    int i, j;
    FILE *f;
    f = fopen("BC.txt", "a");
    for (n = n1; n <= n2; n += 1) {

      for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
          if (i == j)
            graph[i][j] = 0;
          else
            graph[i][j] = rand() % 10;

      for(int i=0;i<n;i++)
        visited[i] = 0;
      cnt = 0;
      dijkstra(0);
      fprintf(f, "%d\t%d\n", n, cnt);
    } //system("gnuplot>load 'command.txt'");
    fclose(f);
 }

void main() {
    int ch;
    printf("1.analysis\t\t2.correctness\t\t0.exit\n");
    for (;;) {
      printf("enter choice: ");
      scanf("%d", &ch);
      switch (ch) {
      case 1:
        analysis();
        break;
      case 2:
        correctness();
        break;
```

```
      case 0:
        printf("exiting..\n");
        exit(0);
      default:
        printf("wrong choice!!\n");
        break;
      }
    }
  }
```

1.

```
set xrange[10:110]
set yrange[0:150]
set xlabel 'N'
set ylabel 'count'
set style data linespoints
plot "BC1.txt" title 'cic bestcase' ,  'WC1.txt' title 'cic worstcase',"BC2.txt" title
'euclid bestcase' ,  'WC2.txt' title 'euclid worstcase',"BC3.txt" title 'rep sub bestcase' ,
'WC3.txt' title 'rep sub worstcase'
pause -1 'hit any key'
```

2.

```
a.
set xrange[10:110]
set yrange[0:110]
set xlabel 'N
set ylabel 'operation count'
set style data linespoints
plot 'BC.txt' title 'bestcase' , 'WC.txt' title 'worstcase'
pause -1 'hit any way'

b.
set xrange[10:110]
set yrange[0:10]
set xlabel 'N
set ylabel 'operation count'
set style data linespoints
plot 'BC.txt' title 'bestcase' , 'WC.txt' title 'worstcase'
pause -1 'hit any way'
```

3.

```
a,b,c.
set xrange[10:110]
set yrange[0:6000]
set xlabel 'N
set ylabel 'operation count'
set style data linespoints
plot 'BC.txt' title 'bestcase' , 'WC.txt' title 'worstcase'
pause -1 'hit any way'
```

4.

```
set xrange[10:110]
set yrange[0:3000]
set xlabel 'N
set ylabel 'operation count'
set style data linespoints
plot 'BC.txt' title 'bestcase' , 'WC.txt' title 'worstcase'
pause -1 'hit any way'
```

5.

```
set xrange[4:1100]
set yrange[0:11000]
set xlabel 'N
set ylabel 'operation count'
set style data linespoints
plot 'BC.txt' title 'bestcase' , 'WC.txt' title 'worstcase'
pause -1 'hit any way'
```

6.

```
set xrange[10:110]
set yrange[0:6000]
set xlabel 'N
set ylabel 'operation count'
set style data linespoints
plot 'BC.txt' title 'bestcase' , 'WC.txt' title 'worstcase'
pause -1 'hit any way'
```

7.

```
set xrange[3:10]
set yrange[0:150]
set xlabel 'N
set ylabel 'operation count'
set style data linespoints
plot 'BC.txt' title 'bestcase' , 'WC.txt' title 'worstcase'
pause -1 'hit any way'
```

8.

```
set xrange[3:11]
set yrange[0:150]
set xlabel 'N
```

	set ylabel 'operation count' set style data linespoints plot 'BC.txt' title 'bestcase' , 'WC.txt' title 'worstcase' pause -1 'hit any way'
9.	set xrange[4:9] set yrange[0:100] set xlabel 'N set ylabel 'operation count' set style data linespoints plot 'BC.txt' title 'analysis' pause -1 'hit any way'
10.	set xrange[4:9] set yrange[0:100] set xlabel 'N set ylabel 'operation count' set style data linespoints plot 'BC.txt' title 'analysis' pause -1 'hit any way'
11.	set xrange[10:110] set yrange[0:700] set xlabel 'N set ylabel 'operation count' set style data linespoints plot 'BC.txt' title 'bestcase' , 'WC.txt' title 'worstcase' pause -1 'hit any way'
12.	a,b. set xrange[4:9] set yrange[0:600] set xlabel 'N set ylabel 'operation count' set style data linespoints plot 'BC.txt' title 'analysis' pause -1 'hit any way'
13.	a,b. set xrange[4:11] set yrange[0:100] set xlabel 'N set ylabel 'operation count' set style data linespoints plot 'BC.txt' title 'analysis' pause -1 'hit any way'
14.	set xrange[4:9] set yrange[0:100] set xlabel 'N set ylabel 'operation count' set style data linespoints plot 'BC.txt' title 'analysis' pause -1 'hit any way'
15.	set xrange[4:9] set yrange[0:100] set xlabel 'N set ylabel 'operation count' set style data linespoints plot 'BC.txt' title 'analysis' pause -1 'hit any way'