**11. Design, develop, code and run the program in any suitable language to implement the Quicksort algorithm. Determine the basis paths and using them derive different test cases, execute these test cases and discuss the test results.**

#include<stdio.h>

1 void quicksort(int x[10],int first,int last)

2 {

3 int temp,pivot,i,j;

4 if(first<last)

5 {

6 pivot=first;

7 i=first;

8 j=last;

9 while(i<j)

10 {

11 while(x[i]<=x[pivot] && i<last)

12 i++;

13 while(x[j]>x[pivot])

14 j--;

15 if(i<j)

16 {

17 temp=x[i];

18 x[i]=x[j];

19 x[j]=temp;

20 }

21 }

22 temp=x[pivot];

x[pivot]=x[j];

x[j]=temp;

quicksort(x,first,j-1);

26 quicksort(x,j+1,last);

27 }

28 }

int main()

{

int a[20],i,key,n;

printf("enter the size of the array max of 20 elements");

scanf("%d",&n);

if(n>0)

{

printf("enter the elements of the array");

for(i=0;i<n;i++)

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

quicksort(a,0,n-1);

printf("the elements in the sorted array is:\n");

for(i=0;i<n;i++)

print f("%d\t",a[i]);

}

else

printf(“size of array is invalid\n”);

}

|  |  |
| --- | --- |
| **NODES in PROGRAM GRAPH** | **NODES in DDPATHS GRAPH** |
| **1** | **A** |
| **2-3** | **B** |
| **4** | **C** |
| **5-8** | **D** |
| **9** | **E** |
| **10** | **F** |
| **11** | **G** |
| **12** | **H** |
| **13** | **I** |
| **14** | **J** |
| **15** | **K** |
| **16-20** | **L** |
| **21** | **M** |
| **22-25** | **N** |
| **26** | **P** |
| **27** | **Q** |
| **28** | **O** |

**Cyclomatic Complexity: V(G) =e-n+p (for closed graph)**

G **= 23-17+ (1)**

**= 7** Test cases

**McCabe’s Basis path method**

Considering DD-Path graph of the program, first we need to find Baseline path. A baseline path consists of maximum number of decision nodes. Using Baseline path we start flipping each decision node for finding new paths.

**Considering Quick Sort program**

Considering DD-Path graph of Quick sort function, function starts at node A and Ends at node O. First, Base

Line path is formed by considering all decision nodes as shown below.

**Baseline Path:** A B **C** D **E** F **G I K** M E N A B C O **P** A B C O.

Flipping each decision node yields

**Flipping at C :** A B C O.

**Flipping at E :** A B C D E N A B C O.

**Flipping at G :** A B C D E F G H G I K M E N A B C O.

**Flipping at I :** A B C D E F G I J I K M E N A B C O.

**Flipping at K :** A B C D E F G I K L M E N A B C O.

**Flipping at P :** A B C D E F G I K L M E N A B C O P A B C O.

**Test Cases for Quick Sort Program**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Cases** | **Description** | **Number of elements (n)** | **Array Elements** | **Comment** |
| TC1 | Enter the basis path consisting of all decision nodes **ABCDEFGIKMENABCOPABCO.** | ---- | Infeasible because path from G to I means no elements in array. | Invalid |
| **TC2** | Enter the basis path consisting of all decision nodes **ABCO.** | 1 | {9} | Valid |
| **TC3** | Enter the basis path consisting of all decision nodes **ABCDENABCO.** | ---- | Path C to D indicates if(first<last) is condition is true. So at first iteration while(x[i]<=x[pivot]&&i<last) condition also should be true and path E to F should be present. But we have EN so Infeasible | Invalid |
| **TC4** | Enter the basis path consisting of all decision nodes **ABCDEFGHGIKMENABCO.** | 2 | {5,4 } | Valid |
| **TC5** | Enter the basis path consisting of all decision nodes **ABCDEFGIJIKMENABCO.** | ---- | Infeasible because path from G to I means no elements in array. | Invalid |
| **TC6** | Enter the basis path consisting of all decision nodes **ABCDEFGIKLMENABCO.** | ---- | Infeasible because path from G to I means no elements in array. | Invalid |
| **TC7** | Enter the basis path consisting of all decision nodes **ABCDEFGIKLMENABCOPABCO.** | ---- | Infeasible because path from G to I means no elements in array. | Invalid |

**Note**

If given array contains a single element, then first=last, if(first<last) condition is true indicates there are more than one elements in the given array. Even when there will be single element While(x[i]<=x[pivot]&&i<last) condition will get executed at least once, because x[i]=x[pivot] is also considered. So path there should be one path G to H present for any feasible solution. So in above table paths containing G to I are all infeasible.