**Assignment # 4**

*20k-0157; Section D.*

**Q1**

char \*SortAlphabeticalArray(char arr[],int len)

{

char temp;

int i, j;

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

for (j = i+1; j < len; j++) {

if (arr[i] > arr[j]) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

return arr;

}

int \*sortNumericArray(int array[],int len)

{

int temp;

int i, j;

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

for (j = i+1; j < len; j++) {

if (array[i] > array[j]) {

temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

}

return array;

}

**Main(If u want to run and check):**

int main(int argc, char \*argv[]) {

int n,k;

char array1[10];

printf("Enter the string : ");

scanf("%s",array1);

n = strlen(array1);

char \*ptr;

ptr=SortAlphabeticalArray(array1,n);

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

{

printf("%c",ptr[k]);

}

int array2[5],e,b,length;

printf("\nEnter integers;\n");

for(e=0;e<=4;e++){

scanf("%d",&array2[e]);

}

int \*a;

a=sortNumericArray(array2,5);

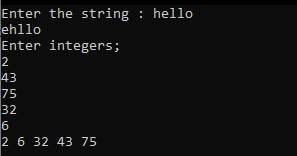
for(b=0;b<=4;b++){

printf("%d ",a[b]);

}

return 0;

}



**Q2**.

void sortAlphabetRecurs(char array[], int n) {

char temp = 'a';

if (n== 1)

return;

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

if (array[i] > array[i+1]) {

temp = array[i];

array[i] = array[i + 1];

array[i + 1] = temp;

}

}

sortAlphabetRecurs(array,(n -1));

}

void sortNumericRecurs(int array[], int n) {

int temp = 0;

if (n== 1)

return;

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

if (array[i]>array[i+1]){

temp = array[i];

array[i] = array[i + 1];

array[i + 1] = temp;

}

}

sortNumericRecurs(array, (n- 1));

}

**Q3**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

/\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/

int strlenVal(char arr[])

{

int i=0,count=0;

while(arr[i]!='\0')

{

count++;

i++;

}

return count;

}

void strlenRef(char arr[],int \*counter)

{

int j=0,k=0;

while(arr[j]!='\0')

{

k++;

j++;

}

\*counter=k;

}

char \*strcatVal(char str1[],char str2[])

{

int c,d,z;

c=strlen(str1);

d=strlen(str2);

for(z=0;z<=d;z++)

{

str1[c]=str2[z];

c++;

}

return str1;

}

void strcatRef(char \*string1,char \*string2,int h,int g)

{

int z,f=0;

f=h+g;

while (\*string1)

{

string1++;

}

for(z=0;z<=f;z++)

{

\*string1=\*string2;

string1++;

string2++;

}

}

char \*strcopyVal(char arr11[],char arr22[])

{

int p;

int lenarr=strlen(arr11);

for(p=0;p<=lenarr;p++)

{

arr22[p]=arr11[p];

}

return arr22;

}

void strcopyRef(char \*arr111,char \*arr222)

{

int u;

while(\*arr111)

{

\*arr222=\*arr111;

arr222++;

arr111++;

}

}

int main(int argc, char \*argv[]) {

// StrLenVal Main

int a;

char array1[10];

printf("Enter string to check length;");

gets(array1);

a=strlenVal(array1);

printf("String length is %d\n",a);

// StrlenRef Main

int \*counter=0;

strlenRef(array1,&counter);

printf("String length by ref is %d\n",counter);

// StrcatVal Main

char str1[50], str2[50];

int res;

printf("\nEnter first string: ");

scanf("%s",str1);

printf("\nEnter second string: ");

scanf("%s",str2);

res=strcatVal(str1,str2);

printf("Concatenated by val; %s\n",res);

//

// StrcatRef Main

char string1[50], string2[50];

int h,g;

printf("\nEnter first string: ");

scanf("%s",string1);

printf("\nEnter second string: ");

scanf("%s",string2);

h=strlen(string1);

g=strlen(string2);

strcatRef(&string1,&string2,h,g);

printf("concatenated by ref ; %s\n",string1);

//

// StrcopyVal Main

char arr11[50],arr22[50];

int o;

printf("\nEnter string: ");

scanf("%s",arr11);

o=strcopyVal(arr11,arr22);

printf("copied string %s\n",o);

//

// StrcopyRef Main

char arr111[50],arr222[50];

printf("\nEnter string: ");

scanf("%s",arr111);

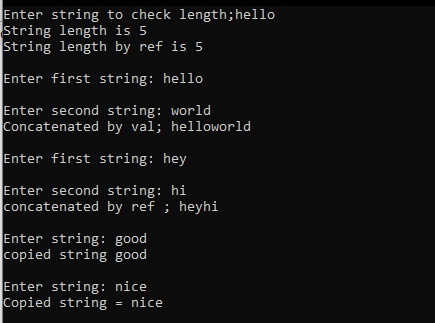
strcopyVal(&arr111,&arr222);

printf("Copied string = %s\n",arr222);

//

return 0;

}



**Q4**

#include <stdio.h>

#include <conio.h>

void qsort(int b[],int left,int right)

{

int i,j,p,tmp,finished,k;

if(right>left)

{

i=left;

j=right;

p=b[left];

finished=0;

while (!finished)

{

do

{

++i;

}

while ((b[i]<=p) && (i<=right));

while ((b[j]>=p) && (j>left))

{

--j;

}

if(j<i)

finished=1;

else

{

tmp=b[i];

b[i]=b[j];

b[j]=tmp;

}

}

tmp=b[left];

b[left]=b[j];

b[j]=tmp;

qsort(b,left,j-1);

qsort(b,i,right);

}

return;

}

int n;

void main()

{

int a[100],i,l,r;

printf("\nENTER THE SIZE OF THE ARRAY: ");

scanf("%d",&n);

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

{

printf("\nENTER#%d: ",i+1);

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

}

printf("\nBEFORE SORTING: \n");

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

{

printf("%5d",a[i]);

}

l=0;

r=n-1;

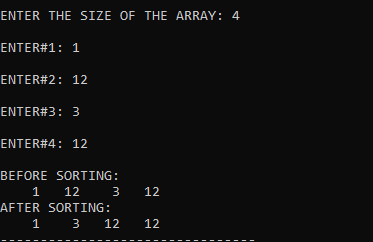
qsort(a,l,r);

printf("\nAFTER SORTING: \n");

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

printf("%5d",a[i]);

}



**Q5**

Recursion is very important in quick sort as the algorithm of quicksort works on breaking the array into subarrays, comparing it to the pivot and then applying the same function on the smaller subarrays .Then those two subarrays are each broken into subarrays, each have a pivot chosen and compared to. Then these subarrays are further broken into more subarrays on which this algorithm is applied to. This function is repeated recursively until the whole array is sorted.

Array on which quicksort is applied;

Pivot choosen;

Array compared to pivot and moved down into subarrays depending that the data at that array position is > or < than the pivot

Sub-Array2

Sub-Array1

Sub-Array2

Sub-Array2

Sub-Array2

Sub-Array1

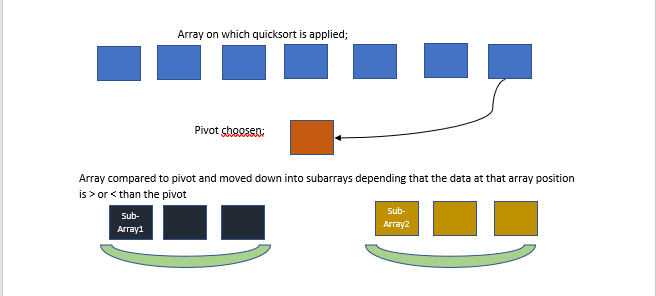
Sub-Array1

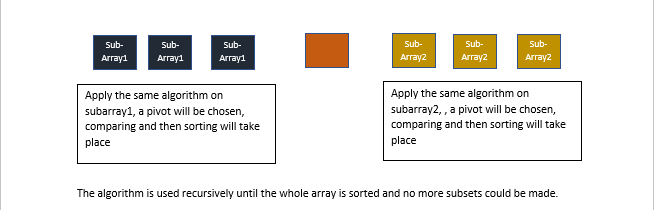
Sub-Array1

Apply the same algorithm on subarray2, , a pivot will be chosen, comparing and then sorting will take place

Apply the same algorithm on subarray1, a pivot will be chosen, comparing and then sorting will take place

The algorithm is used recursively until the whole array is sorted and no more subsets could be made.





**Q6.**

#include <stdio.h>

// A recursive binary search function. It returns location of x in

// given array arr[l..r] is present, otherwise -1

int binarySearch(int arr[], int key,int start, int end)

{

if (end >= start)

{

int mid = start + (end - start)/2;

if (arr[mid] == key)

return mid;

if (arr[mid] > key)

return binarySearch(arr,key, start, mid-1);

return binarySearch(arr,key, mid+1, end);

}

return -1;

}

int main(void)

{

int arr[5],x,i;

printf("Enter array; \n");

for(i=0;i<=4;i++)

{

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

}

int n = sizeof(arr)/ sizeof(arr[0]);

printf("Enter integer to search for; ");

scanf("%d",&x);

int result = binarySearch(arr,x, 0, n-1);

if(result == -1)

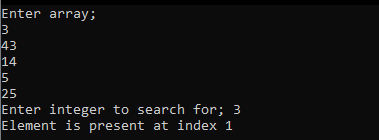
printf("Element is not present in array");

else

printf("Element is present at index %d", result+1);

return 0;

}



**Q7**

Like a stack of plates which are kept one over another and are retrieved from the top, Stacks are used in memory management by the operating system and for backtracking if we want to find the path of some data. Last in first out.

Like a queue of customers standing in line with the first one in line getting his work down first, Queue is used in CPU and task scheduling, that is giving the CPU power to the first task that requests it, while the rest wait in queue. First in first out. Similarly, in printers where the task sent first is printed first while the others are queued.

**Q8**

#include <stdio.h>

#include <stdlib.h>

/\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/

int x;

void revhalfpyramid(int n)

{

int a;

if(n==0)

return;

if(n!=0)

{

a=n;

while(a!=0)

{

printf("\* ");

a--;

}

printf("\n");

revhalfpyramid(n-1);

}

}

void square(int n)

{

int b=0;

if(n==0)

return;

if(n!=0)

{

while(b!=x)

{

printf("\* ");

b++;

}

printf("\n");

square(n-1);

}

}

int main(int argc, char \*argv[]) {

int n;

printf("Enter length n of the shape; ");

scanf("%d",&n);

x=n;

revhalfpyramid(n);

printf("\n");

square(n);

return 0;

}

