Assignment

1.

Performance of Quick-Sort with four different pivots for partitioning

Pivot-1: Choose the first element as pivot

Pivot-2: Choose randomly the pivot element

Pivot-3: Median of {First element, Middle element, Last element}

Pivot-4: Median of {n/4th element, middle element, 3n/4th element}

Performance of Merge-Sort

Input Array types:

1. Random numbers of integers in the range 0 to 10k ( k = 4,5,6,7)

2. Sorted array of (1)

3. Almost Sorted of (1). ( This can be obtained by swapping 1% pairs of the

sorted array)

Implement Quick\_Sort(Array, Size\_of\_Array, Pivot\_type)

Source\_Code:

#include<stdio.h>

#define MAX 100

#include<stdlib.h>

#include<time.h>

#include<math.h>

int \*swapFun(int \*randArray, int n, int swapCount)

{

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

{

int temp = randArray[i];

randArray[i] = randArray[i+1];

randArray[i+1] = temp;

}

return randArray;

}

//Bubble sort for input in sorted order

int \*bubble\_sort(int \*randArray, int n)

{

int a;

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

{

for (int j = i + 1; j < n; ++j)

{

if (randArray[i] > randArray[j])

{

a = randArray[i];

randArray[i] = randArray[j];

randArray[j] = a;

}

}

}

return randArray;

}

//Create a random array of size n

int \*randomArray(int n, int \*randArray, int kVal )

{

int j;

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

randArray[j]=rand()% kVal;

/\*printf("\n Elements of the array::");

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

{

printf("\n Element number %d :%d",i+1,randArray[i]);

} \*/

return randArray;

}

//Median of {First element, Middle element, Last element}

int \*medianFun(int arr[], int left, int right, int \*medianArray)

{

int mid = (left+right-1)/2;

int a = arr[left];

int b = arr[mid];

int c = arr[right];

int pos, value;

if(a <= b && b <= c)

{

value = b;

pos = mid;

}

else if(c <= b && b <= a)

{

value = b;

pos = mid;

}

else if(a <= c && c <= b)

{

value = c;

pos = right;

}

else if(b <= c && c <= a)

{

value = c;

pos= right;

}

else

{

value = a;

pos = left;

}

medianArray[0] = value;

medianArray[1] = pos;

return medianArray;

}

//Median of {n/4th element, middle element, 3n/4th element}

int \*medianFun1(int arr[], int left, int right, int \*medianArray)

{

int mid = (left+right-1)/2;

int n4 = (right-left)/4 + left;

int n34 = (3\*(right-left))/4+ left;

int a = arr[n4];

int b = arr[mid];

int c = arr[n34];

int pos, value;

if(a <= b && b <= c)

{

value = b;

pos = mid;

}

else if(c <= b && b <= a)

{

value = b;

pos = mid;

}

else if(a <= c && c <= b)

{

value = c;

pos = n34;

}

else if(b <= c && c <= a)

{

value = c;

pos= n34;

}

else

{

value = a;

pos = n4;

}

medianArray[0] = value;

medianArray[1] = pos;

return medianArray;

}

int \*pivotFun(int arr[], int left, int right, int pivotChoice, int \*pivotArray){

int medianArray[5], \*medianPointer;

//pivot choice --> Left

if(pivotChoice == 1)

{

pivotArray[0]= arr[left];

pivotArray[1] = left;

}

//pivot choice --> Random

else if(pivotChoice == 2)

{

int randPos = (rand() % (right - left + 1)) + left;

pivotArray[0] = arr[randPos];

pivotArray[1] = randPos;

}

//pivot choice -->median

else if(pivotChoice == 3)

{

medianPointer = medianFun(arr, left, right, medianArray);

pivotArray[0] = medianPointer[0];//element

pivotArray[1] = medianPointer[1];//position

}

else

{

medianPointer = medianFun1(arr, left, right, medianArray);

pivotArray[0] = medianPointer[0];//element

pivotArray[1] = medianPointer[1];//position

}

return pivotArray;

}

int partition(int arr[], int left, int right, int pivotChoice)

{

int temp,i,j,pivot;

int \*pivArray;

int pivotArray[5];

i = left+1;

j = right;

pivArray = pivotFun(arr,left, right, pivotChoice, pivotArray);

int pivotPos = pivArray[1];

//Swap of elements left and position element

int temp1 = arr[pivotPos];

arr[pivotPos] = arr[left];

arr[left] = temp1;

pivot = arr[left];

while(i <= j)

{

while(arr[i] < pivot && i<right)

i++;

while(arr[j] > pivot)

j--;

if(i < j)

{

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

i++;

j--;

}

else

i++;

}

arr[left]=arr[j];

arr[j]=pivot;

return j;

}/\*End of partition()\*/

//quick sort function

void quick\_sort(int \*arr,int left, int right, int pivotChoice)

{

int pivloc;

//base condition

if(left >= right)

return;

pivloc = partition(arr,left, right, pivotChoice);

quick\_sort(arr, left, pivloc-1, pivotChoice); /\*process left sublist\*/

quick\_sort(arr, pivloc+1, right, pivotChoice); /\*process right sublist\*/

}/\*End of quick()\*/

int main()

{

int arr[MAX],n,i,inp,k;

while(1)

{

printf("\n Choose Input");

printf("\n 1. Random Integers in the range of 10^k, k between 4 and 7 \n 2. Sorted array \n 3. Almost sorted array \n 4. Exit \n");

scanf("%d", &inp);

printf("Enter the size of array: ");

scanf("%d",&n);

printf("\n Enter the value of k in 10^k");

scanf("%d",&k);

int val = pow(10,k);

int \*randArrayPointer;

int randArray[n];

//Random integer input

if(inp ==1)

{

randArrayPointer = randomArray(n, randArray,val);

/\* for(i=0;i<n;i++)

{

printf("\n Main\_Element number %d :%d",i+1,randArrayPointer[i]);

} \*/

}//End of inp1

//Sorted array input

else if (inp ==2)

{

randArrayPointer = randomArray(n, randArray,val);

randArrayPointer = bubble\_sort(randArrayPointer, n);

/\*for(i=0;i<n;i++)

{

printf("\n Main\_inp2\_Element number %d :%d",i+1,randArrayPointer[i]);

}

\*/

}

//Almost Sorted array input

else if (inp ==3)

{

randArrayPointer = randomArray(n, randArray, val);

randArrayPointer = bubble\_sort(randArrayPointer, n);

int pairCount = (n\*(n-1))/2; //nC2 pairs

int swapCount =(int)( 0.01 \* (float)pairCount);

printf("\n Swap Count %d",swapCount);

randArrayPointer = swapFun(randArrayPointer, n,swapCount);

/\*for(i=0;i<n;i++)

{

printf("\n Main\_inp3\_Element number %d :%d",i+1,randArrayPointer[i]);

}\*/

}

else if(inp == 4)

{

break;

}

int pivotChoice;

printf("\n Enter pivot choice\n 1. First \n 2. Random \n 3. Median first, middle, last \n 4. Median n/4, middle, 3n/4 \n");

scanf("%d",&pivotChoice);

clock\_t t;

t = clock();

quick\_sort(randArrayPointer, 0, n-1,pivotChoice);

t = clock() - t;

double time\_taken = ((double)t)/CLOCKS\_PER\_SEC; // in seconds

printf("\n quick\_sort() took %f seconds to execute \n", time\_taken);

/\*printf(" Sorted list is :\n");

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

printf("%d ",randArrayPointer[i]);

printf("\n");

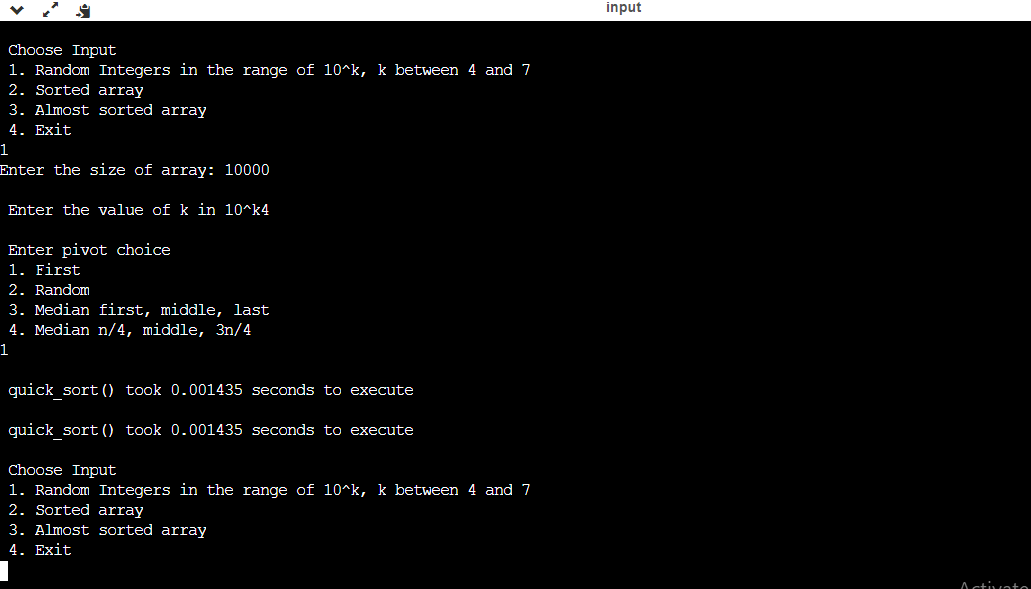
\*/

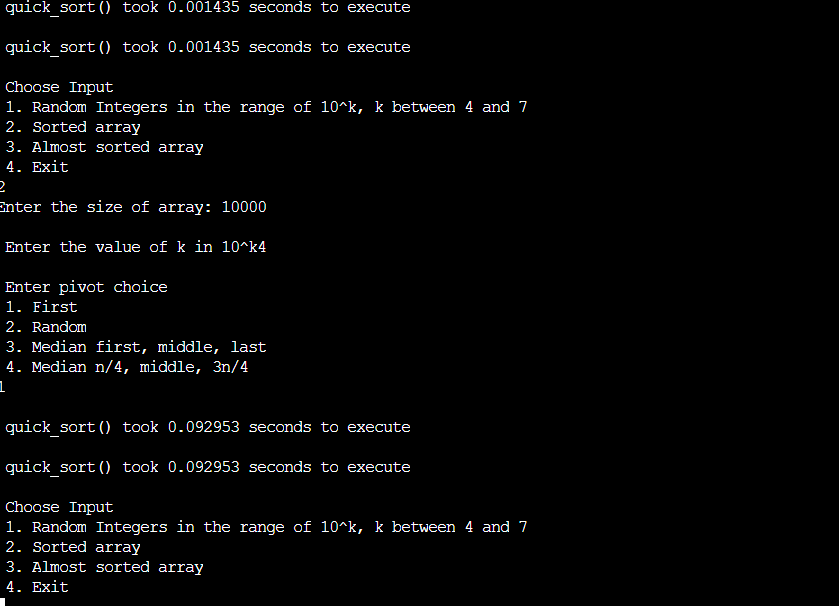
printf("\n quick\_sort() took %f seconds to execute \n", time\_taken);

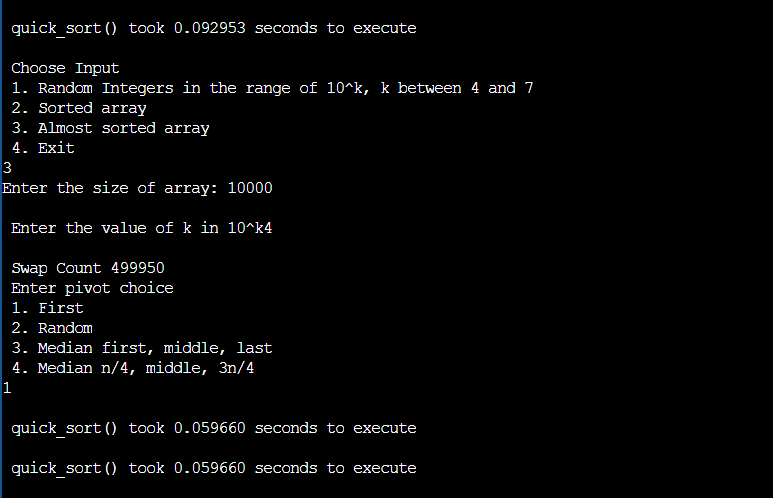
}//End of while

}

Output:







2.

Implement Merge\_Sort(Array, Size\_of\_Array)

Source\_Code:

#include<stdio.h>

#define MAX 100

#include<stdlib.h>

#include<time.h>

#include<math.h>

int \*swapFun(int \*randArray, int n, int swapCount)

{

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

{

int temp = randArray[i];

randArray[i] = randArray[i+1];

randArray[i+1] = temp;

}

return randArray;

}

int \*bubble\_sort(int \*randArray, int n)

{

int a;

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

{

for (int j = i + 1; j < n; ++j)

{

if (randArray[i] > randArray[j])

{

a = randArray[i];

randArray[i] = randArray[j];

randArray[j] = a;

}

}

}

return randArray;

}

int \*randomArray(int n, int \*randArray,int kVal)

{

int j;

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

randArray[j]=rand()%kVal;

printf("\n Elements of the array::");

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

{

printf("\n Element number %d :%d",i+1,randArray[i]);

}

return randArray;

}

//Merges arr[left1:right1] and arr[left2:right2] to temp[left1:right2]

void merge( int arr[], int temp[], int left1, int right1, int left2, int right2 )

{

int i = left1;

int j = left2 ;

int k = left1 ;

while( (i <= right1) && (j <=right2))

temp[k++] = (arr[i]<=arr[j])? arr[i++]: arr[j++];

while( i <= right1 )//If any elements were left on left side of array

temp[k++]=arr[i++];

while( j <= right2 )//If any elements were left on right side of array

temp[k++]=arr[j++];

}

//After completing every division of problem, we keep writing over and over again,atlast we will have sorted order.

void copy(int \*arr, int \*temp, int left, int right )

{

for(int i=left; i<=right; i++)

arr[i]=temp[i];

}

void merge\_sort(int \*arr, int left, int right)

{

int mid;

int temp[MAX];

if(left<right)//if more than one element

{

mid = (left+right)/2; //Finding the mid,it would be useful for dividing

merge\_sort( arr, left , mid ); //Sort array from low to mid

merge\_sort( arr, mid+1, right ); //Sort array from mid+1 to right

//Merge arr[low:mid] and arr[mid+1:right] to temp[left:right]

merge( arr, temp, left, mid, mid+1, right );

//Copy temp[left:right] to arr[left:right]

copy(arr,temp,left, right);

}

}

int main()

{

int arr[MAX],n,i,inp,k;

while(1)

{

printf("\n Choose Input");

printf("\n 1. Random Integers in the range of 10^k, k between 4 and 7 \n 2. Sorted array \n 3. Almost sorted array \n 4. Exit \n");

scanf("%d", &inp);

printf("\n Enter the value of k in 10^k");

scanf("%d",&k);

printf("Enter the size of array: ");

scanf("%d",&n);

int \*randArrayPointer;

int randArray[n];

int val = pow(10,k);

if(inp ==1)

{

randArrayPointer = randomArray(n, randArray,val);

/\* for(i=0;i<n;i++)

{

printf("\n Main\_Element number %d :%d",i+1,randArrayPointer[i]);

} \*/

}//End of inp1

else if (inp == 2)

{

randArrayPointer = randomArray(n, randArray,val);

randArrayPointer = bubble\_sort(randArrayPointer, n);

/\*for(i=0;i<n;i++)

{

printf("\n Main\_inp2\_Element number %d :%d",i+1,randArrayPointer[i]);

}

\*/

}

else if (inp ==3)

{

randArrayPointer = randomArray(n, randArray, val);

randArrayPointer = bubble\_sort(randArrayPointer, n);

int pairCount = (n\*(n-1))/2;

int swapCount =(int)( 0.01 \* (float)pairCount);

printf("\n Swap Count %d",swapCount);

randArrayPointer = swapFun(randArrayPointer, n,swapCount);

/\*for(i=0;i<n;i++)

{

printf("\n Main\_inp3\_Element number %d :%d",i+1,randArrayPointer[i]);

}\*/

}

else if(inp == 4)

{

break;

}

int l=0,r=n-1;

clock\_t t;

t = clock();

merge\_sort( randArrayPointer, l, r);

t = clock() - t;

double time\_taken = ((double)t)/CLOCKS\_PER\_SEC; // in seconds

printf("\n merge\_sort() took %f seconds to execute \n", time\_taken);

printf(" Sorted list is :\n");

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

printf("%d ",randArrayPointer[i]);

printf("\n");

printf("\n merge\_sort() took %f seconds to execute \n", time\_taken);

}//End of while

}

Output:

