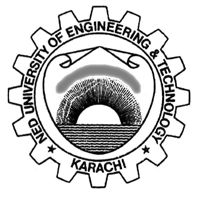
**Data Structure Algorithm and Application CT-157**

Workbook

**Data Structure Algorithm and Application**

**CT-157**



**Name: Muhammad Shaheer Akram**

**Roll No: CT-055**

**Semester: Second**

**Year: First**

**Department: Computer Science and Information Technology**

LAB # 1

**OBJECT:** To insert and delete in Word Processing

**THEORY**:

Suppose in given text T. We want to insert string P so that P begins in position R I text T. This operation in denoted as

INSERT (T, R, P)

For example, if T = “I am student of second year science”

R=30, and P=” computer”. Then INSERT (T, R, P) will result in the string. “I am student of second year computer science”.

Suppose in a given text T. We want to delete the substring that begins in position R in text T and has length L. This operation is denoted as

DELETE (T, R, L)

For example, if T=ABCDEFGH, R=4 & L=3

Then DELETE (T, R, L) will result in string “ABCGH”

**Algorithm A:**

INSERT (T, R, P)

This algorithm uses two strings temp 1, temp 2

1. Temp 1 = substring (T, 1, R-1)
2. Temp 2= substring (T, R, Length(T)-R+1)
3. Concatenate (Temp 1, P)
4. Concatenate (Temp 1, Temp 2)
5. T= Temp 1

**Algorithm B:**

DELETE (T, R, L)

This algorithm uses temporary strings Temp 1 and Temp 2

1. Temp 1 = substring (T, 1, R-1)
2. Temp 2 = substring (T, R+L, Length(T)-R-L+1)
3. T = Concatenate (temp 1, temp 2)
4. Exit

**SOURCE CODE FOR ALGORITHM A:**

#include <stdio.h>

#include <conio.h>

#include <string.h>

int main(){

char a[10],b[10],c[10];

int p=0,r=0,i=0,t=0;

int x,g,s,n,o;

puts("Enter First String:");

gets(a);

puts("Enter Second String:");

gets(b);

printf("Enter the position where the item has to be inserted: ");

scanf("%d",&p);

r = strlen(a);

n = strlen(b);

// Copying the input string into another array

while(i <= r){

c[i]=a[i];

i++;

}

s = n+r;

o = p+n;

//Adding the sub-string

for(i=p;i<s;i++){

x = c[i];

if(t<n){

a[i] = b[t];

t=t+1;

}

a[o]=x;

o=o+1;

}

printf("%s", a);

return 0;

}

**SOURCE CODE FOR ALGORITHM B:**

#include <stdio.h>

#include <conio.h>

#include <string.h>

void delchar(char \*x,int a, int b);

int main(){

char string[10];

int n,pos,p;

puts("Enter a string :");

gets(string);

printf("Enter the position from where you want to delete:");

scanf("%d",&pos);

printf("Enter the number of characters to be deleted :");

scanf("%d",&n);

delchar(string, n,pos);

return 0;

}

void delchar(char \*x,int a, int b){

if ((a+b-1) <= strlen(x)){

strcpy(&x[b-1],&x[a+b-1]);

puts(x);

}

}

LAB # 2

**OBJECT:** To implement quick sort

**EXPLANATION:** Quick sort is one of the most popular sorting algorithms based on divide et

impera strategy, resulting in an *O*(*n log n*) complexity. The algorithm starts by picking an item, called pivot, and moving all smaller items before it, while all greater elements after it. This is the main quick sort operation, called partition, recursively repeated on lesser and greater sub lists until their size is one or zero - in which case the list is implicitly sorted.

Choosing an appropriate pivot, as for example the median element is fundamental for avoiding the drastically reduced performance of *O*(*n*2).

**PSEUDOCODE:**

**PSEUDOCODE FOR RECURSIVE QUICK SORT FUNCTION**

quicksort(arr[], low, high){

if(low<high){

pi = partition(arr, low, high)

quicksort(arr, low, pi-1);

quicksort(arr, pi+1, high);

}

}

**PSEUDOCODE FOR PARTITION**

partition(arr[], low, high){

pivot = arr[high];

i = (low-1);

for (j=low; j<=high-1; j++){

if(arr[j] <= pivot){

i++;

swap arr[i] and arr[j]

}

}

Swap arr[i+1] and arr[high]

return (i+1)

}

**ALGORITHM:**

1)algorithm QuickSort(list)

2) Pre: list ;

3) Post: list has been sorted into values of ascending order

4) if list.Count = 1 // already sorted

5) return list

6) end if

7) pivot MedianValue(list)

8) for i 0 to list.Count¡1

9) if list[i] = pivot

10) equal.Insert(list[i])

11) end if

12) if list[i] < pivot

13) less.Insert(list[i])

14) end if

15) if list[i] > pivot

16) greater.Insert(list[i])

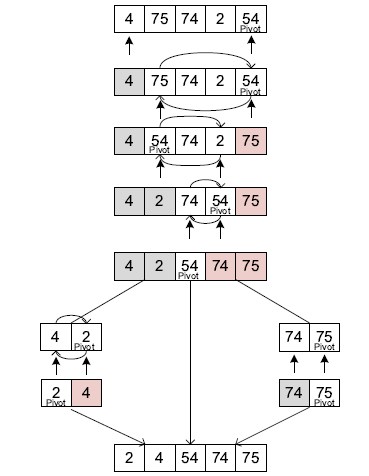
17) end if

18) end for

19) return Concatenate(QuickSort(less), equal, QuickSort(greater))

20) end Quicksort

**ILLUSTRATION:**



**SOURCE CODE:**

#include<stdio.h>

int main(){

int arr[] = {4, 75, 74, 2, 54};

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

quickSort(arr, 0, n-1);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}

void swap(int\* a, int\* b){

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition(int arr[], int low, int high){

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high- 1; j++){

if (arr[j] <= pivot){

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high){

if (low < high){

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void printArray(int arr[], int size){

int i;

for (i=0; i < size; i++){

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

printf("\n");

}

}

LAB # 3

**OBJECT:** To implement linear search

**EXPLANATION:** linear search is a very simple search algorithm. In this type of search, a sequential search is made overall items one by one. Every item is checked and if a match is found then that particular item is returned otherwise the search continues till the end of the data collection

**ALGORITHM:**

Linear Search (Array A, Value x)

Step 1: **Set** i to 1

Step 2: **If** i>n then go to step 7

Step 3: **If** A[i] = x then go to step 6

Step 4: **Set** i to i+1

Step 5: Go to step 2

Step 6: print element n found at index i go to step 8

Step 7: print element not found

Step 8: **Exit**

**PSEUDOCODE:**

**procedure** linear\_search(list, value)

**for** each item in the list

**if** match, item = value

**return** item’s location

**end** if

**end** for

**end** procedure

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

int main(){

int arr[100],size,search,flag,i;

printf("Enter size of array : ");

scanf("%d",&size);

printf("Enter elements of array : ");

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

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

flag=0;

printf("Enter element to search : ");

scanf("%d",&search);

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

{

if(arr[i]==search)

{

flag=1;

break;

}

}

if(flag==1)

printf("%d is found at position %d ",search,i+1);

else

printf("We were unable to find your number!!");

return 0;

}

LAB # 4

**OBJECT:** To implement bubble sort

**EXPLANATION:** Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

**ALGORITHM:**

1) **algorithm** BubbleSort(list)

2) **Pre**: list ;

3) **Post**: list has been sorted into values of ascending order

4) **for** i 0 to list:Count ¡ 1

5) **for** j 0 to list:Count ¡ 1

6) **if** list[i] < list[j]

7) Swap(list[i]; list[j])

8) **end** **if**

9) **end** **for**

10) **end for**

11) **return** list

12) **end** BubbleSort

**ILLUSTRATION:**

**First Pass:**

( **5** **1** 4 2 8 ) –> ( **1** **5** 4 2 8 ), Here, algorithm compares the first two elements, and swaps since 5 > 1.  
( 1 **5** **4** 2 8 ) –>  ( 1 **4** **5** 2 8 ), Swap since 5 > 4  
( 1 4 **5** **2** 8 ) –>  ( 1 4 **2** **5** 8 ), Swap since 5 > 2  
( 1 4 2 **5** **8** ) –> ( 1 4 2 **5** **8** ), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

**Second Pass:**

( **1** **4** 2 5 8 ) –> ( **1** **4** 2 5 8 )  
( 1 **4** **2** 5 8 ) –> ( 1 **2** **4** 5 8 ), Swap since 4 > 2  
( 1 2 **4** **5** 8 ) –> ( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) –>  ( 1 2 4 **5** **8** )  
Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

**Third Pass:**  
( **1** **2** 4 5 8 ) –> ( **1** **2** 4 5 8 )  
( 1 **2** **4** 5 8 ) –> ( 1 **2** **4** 5 8 )  
( 1 2 **4** **5** 8 ) –> ( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) –> ( 1 2 4 **5** **8** )

**SOURCE CODE:**

#include <stdio.h>

int main(){

int arr[] = {33, 22, 44, 11, 55};

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

bubbleSort(arr, n);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}

void swap(int \*xp, int \*yp){

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void bubbleSort(int arr[], int n){

int i, j;

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

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

if (arr[j] > arr[j+1])

swap(&arr[j], &arr[j+1]);

}

}

}

void printArray(int arr[], int size){

int i;

for (i=0; i < size; i++){

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

printf("\n");

}

}

LAB # 5

**OBJECT:** C Program to implement Linear Search

**EXPLANATION:** This C Program implements linear search. Linear Search is also called sequential search linear search is a method for finding an element in a list, that consists of checking each element in a list, one at a time in a sequence, until the desired one is found.

**ALGORITHM:**

Step 1: Set i to 1

Step 2: if i > n then go to step 7

Step 3: if A[i] = x then go to step 6

Step 4: Set i to i + 1

Step 5: Go to Step 2

Step 6: Print Element x Found at index i and go to step 8

Step 7: Print element not found

Step 8: Exit

**Pseudocode:**

for each item in the list

if match item == value

return the item's location

end if

end for

**Source code:**

#include<stdio.h>

int main(){

int array[10];

int i,num,keynum,found=0;

printf("enter the values of num");

scanf("%d",&num);

printf("enter values one by one");

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

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

printf("input array\n");

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

printf("%d\n",array[i]);

printf("enter the element to be searched");

scanf("%d",&keynum);

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

if(array[i]==keynum)

{

found=1;

break;

}

if(found==1)

printf("element found");

else

printf("element not found");

}