**Program – 7**

**AIM – Write an algorithm and program to sort n numbers of elements using Quick Sort.**

**Algorithm –**

quickSort(arr[], low, high)

{

if (low < high)

{

pivot\_index = partition(arr, low, high);

quickSort(arr, low, pivot\_index - 1);

quickSort(arr, pivot\_index + 1, high);

}

}

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], arr[j]);

}

}

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

return (i + 1);

}

**Using arrays**

**Source Code -**

#include<iostream>

#include<cstdlib>

using namespace std;

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

{

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

int Partition(int a[], int low, int high)

{

int pivot, index, i;

index = low;

pivot = high;

for(i=low; i< high; i++)

{

if(a[i] < a[pivot])

{

swap(&a[i], &a[index]);

index++;

}

}

swap(&a[pivot], &a[index]);

return index;

}

intRandomPivotPartition(int a[], int low, int high)

{

intpvt, n, temp;

n = rand();

pvt = low + n%(high-low+1);

swap(&a[high], &a[pvt]);

return Partition(a, low, high);

}

intQuickSort(int a[], int low, int high)

{

intpindex;

if(low < high)

{

pindex = RandomPivotPartition(a, low, high);

QuickSort(a, low, pindex-1);

QuickSort(a, pindex+1, high);

}

return 0;

}

int main()

{

int n, i;

cout<<"\nEnter the number of data elements to be sorted: ";

cin>>n;

intarr[n];

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

{ cout<<"Enter element "<<i+1<<": ";

cin>>arr[i];

}

QuickSort(arr, 0, n-1);

cout<<"\nSorted Data ";

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

cout<<"->"<<arr[i];

return 0;

}

**Using Linked List**

**Source Code –**

#include <iostream>

#include <cstdio>

using namespace std;

struct Node

{ int data;

struct Node \*next;

};

void push(struct Node\*\* head\_ref, intnew\_data)

{ struct Node\* new\_node = new Node;

new\_node->data = new\_data;

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

voidprintList(struct Node \*node)

{

while (node != NULL)

{

printf("%d ", node->data);

node = node->next;

}

printf("\n");

}

struct Node \*getTail(struct Node \*cur)

{

while (cur != NULL && cur->next != NULL)

cur = cur->next;

return cur;

}

struct Node \*partition(struct Node \*head, struct Node \*end,

struct Node \*\*newHead, struct Node \*\*newEnd)

{

struct Node \*pivot = end;

struct Node \*prev = NULL, \*cur = head, \*tail = pivot;

while (cur != pivot)

{

if (cur->data < pivot->data)

{

// First node that has a value less than the pivot - becomes

// the new head

if ((\*newHead) == NULL)

(\*newHead) = cur;

prev = cur;

cur = cur->next;

}

Else

{

if (prev)

prev->next = cur->next;

struct Node \*tmp = cur->next;

cur->next = NULL;

tail->next = cur;

tail = cur;

cur = tmp;

}

}

if ((\*newHead) == NULL)

(\*newHead) = pivot;

(\*newEnd) = tail;

return pivot;

}

struct Node \*quickSortRecur(struct Node \*head, struct Node \*end)

{

if (!head || head == end)

return head;

Node \*newHead = NULL, \*newEnd = NULL;

struct Node \*pivot = partition(head, end, &newHead, &newEnd);

if (newHead != pivot)

{

struct Node \*tmp = newHead;

while (tmp->next != pivot)

tmp = tmp->next;

tmp->next = NULL;

newHead = quickSortRecur(newHead, tmp);

tmp = getTail(newHead);

tmp->next = pivot;

}

pivot->next = quickSortRecur(pivot->next, newEnd);

returnnewHead;

}

voidquickSort(struct Node \*\*headRef)

{

(\*headRef) = quickSortRecur(\*headRef, getTail(\*headRef));

return;

}

int main()

{

struct Node \*a = NULL;

push(&a, 5);

push(&a, 20);

push(&a, 4);

push(&a, 3);

push(&a, 30);

cout<< "Linked List before sorting \n";

printList(a);

quickSort(&a);

cout<< "Linked List after sorting \n";

printList(a);

return 0;

}

**Using Linked List and without recursion**

**Source Code –**

#include <iostream>

#include <cstdio>

using namespace std;

structLLNode {

int data;

LLNode \*next;

};

voidinsertAtBeginning(LLNode\*\*head,intdataToBeInserted)

{

LLNode \*curr = new LLNode;

curr->data = dataToBeInserted;

curr->next = NULL;

if(\*head == NULL) {

\*head=curr;

}

else {

curr->next = \*head;

\*head = curr;

}

}

void display(LLNode\*\*head)

{

LLNode \*temp = \*head;

while(temp!=NULL) {

if(temp->next!=NULL)

cout<<temp->data<<" ->";

else

cout<<temp->data;

temp=temp->next

}

cout<<endl;

}

LLNode \*getTail(LLNode \*cur)

{

while (cur != NULL && cur->next != NULL)

cur = cur->next;

return cur;

}

LLNode \*partition(LLNode \*head, LLNode \*end, LLNode \*\*newHead, LLNode \*\*newEnd)

{

LLNode \*pivot = end;

LLNode \*prev = NULL, \*cur = head, \*tail = pivot;

while(cur != pivot) {

if(cur->data < pivot->data) {

if((\*newHead) == NULL)

(\*newHead) = cur;

prev = cur;

cur = cur->next;

}

else { if(prev)

prev->next = cur->next

LLNode \*tmp = cur->next;

cur->next = NULL;

tail->next = cur;

tail = cur;

cur = tmp;

}

}

if((\*newHead) == NULL)

(\*newHead) = pivot;

(\*newEnd) = tail;

return pivot;

}

LLNode \*quickSortRecur(LLNode \*head, LLNode \*end)

{

if (!head || head == end)

return head;

LLNode \*newHead = NULL, \*newEnd = NULL;

LLNode \*pivot = partition(head, end, &newHead, &newEnd);

if (newHead != pivot) {

LLNode \*tmp = newHead;

while (tmp->next != pivot)

tmp = tmp->next;

tmp->next = NULL;

newHead = quickSortRecur(newHead, tmp);

tmp = getTail(newHead);

tmp->next = pivot;

}

pivot->next = quickSortRecur(pivot->next, newEnd);

returnnewHead;

}

voidquickSort(LLNode \*\*headRef)

{ (\*headRef) = quickSortRecur(\*headRef, getTail(\*headRef));

return;

}

int main() {

LLNode \*head = NULL;

LLNode \*tail = NULL;

insertAtBeginning(&head, 6);

insertAtBeginning(&head, 16);

insertAtBeginning(&head, 15);

insertAtBeginning(&head, 50);

insertAtBeginning(&head, 1);

insertAtBeginning(&head, 23);

cout<< "Linked List before sorting \n";

display(&head);

quickSort(&head);

cout<< "Linked List after sorting \n";

display(&head);

return 0;

}