LAB-8

Sorting and Merging Operations

Q1) Implement selection sort on an array of integers and print the array after each iteration to show the sorting progress.

```
#include <stdio.h>

int main(){
   int n,i,j,minIndex,temp;
   printf("Enter size: ");
   scanf("%d",&n);
   int arr[n];
   for(i=0;i<n;i++){
      scanf("%d",&arr[i]);
   }
   for(i=0;i<n-1;i++){
      minIndex=i;
      for(j=i+1;j<n;j++){
         if(arr[j]<arr[minIndex]){
            minIndex=j;
         }
      }
      temp=arr[i];
      arr[i]=arr[minIndex];
      arr[minIndex]=temp;
      printf("After iteration %d: ",i+1);
      for(j=0;j<n;j++){
         printf("%d ",arr[j]);
      }
      printf("\n");
   }
   return 0;
}
```

Sample Input & Output

```
Enter size: 4
20
```

```
2034

3420

34

After iteration 1: 20 2034 3420 34

After iteration 2: 20 34 3420 2034

After iteration 3: 20 34 2034 3420
```

Q2) Implement selection sort on an array of integers and print the array after each iteration to show the sorting progress.

```c
#include <stdio.h>

void selectionSort(int arr[],int n){
   int i,j,minIndex,temp;
   for(i=0;i<n-1;i++){
      minIndex=i;
      for(j=i+1;j<n;j++){
         if(arr[j]<arr[minIndex]){
            minIndex=j;
         }
      }
      temp=arr[i];
      arr[i]=arr[minIndex];
      arr[minIndex]=temp;
      printf("After iteration %d: ",i+1);
      for(j=0;j<n;j++){
         printf("%d ",arr[j]);
      }
      printf("\n");
   }
}

int main(){
   int n;
   printf("Enter size: ");
   scanf("%d",&n);
   int arr[n];
```

```
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    selectionSort(arr,n);
    return 0;
}
```

Sample Input & Output

```
Enter size: 4

2034

3420

20

34

After iteration 1: 20 3420 2034 34

After iteration 2: 20 34 2034 3420

After iteration 3: 20 34 2034 3420
```

Q3) Design a function that performs partial selection sort — it only sorts the first k smallest elements.

```
#include <stdio.h>

void partialSelectionSort(int arr[],int n,int k){
    int i,j,minIndex,temp;
    for(i=0;i<k && i<n-1;i++){
        minIndex=i;
        for(j=i+1;j<n;j++){
            if(arr[j]<arr[minIndex]){
                minIndex=j;
            }
        }
        temp=arr[i];
        arr[i]=arr[minIndex];
        arr[minIndex]=temp;
        printf("After iteration %d: ",i+1);
        for(j=0;j<n;j++){
            printf("%d ",arr[j]);
        }
        printf("\n");
    }
}

int main(){
    int n,k;
    printf("Enter size: ");
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    printf("Enter k: ");
    scanf("%d",&k);
    partialSelectionSort(arr,n,k);
    printf("Final array: ");
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
    printf("\n");
    return 0;
}
```

Sample Input & Output

```
Enter size: 4
2034
3420
20
34
Enter k: 2
After iteration 1: 20 3420 2034 34
After iteration 2: 20 34 2034 3420
Final array: 20 34 2034 3420
```

Q4) Implement selection sort on a singly linked list using node swaps.

```
#include <stdio.h>
#include <stdlib.h>

struct Node{
   int data;
   struct Node *next;
};

struct Node* createNode(int value){
   struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
   newNode->data=value;
   newNode->next=NULL;
   return newNode;
}

void append(struct Node** head,int value){
   struct Node* newNode=createNode(value);
   if(*head==NULL){
      *head=newNode;
      return;
   }
   struct Node* temp=*head;
   while(temp->next!=NULL){
      temp=temp->next;
   }
   temp->next=newNode;
}

void display(struct Node* head){
   struct Node* temp=head;
   while(temp!=NULL){
      printf("%d ",temp->data);
      temp=temp->next;
   }
   printf("\n");
}

void selectionSort(struct Node* head){
   struct Node* current=head;
   while(current!=NULL){
      struct Node* minNode=current;
      struct Node* nextNode=current->next;
```

```
        while(nextNode!=NULL){
            if(nextNode->data<minNode->data){
                minNode=nextNode;
            }
            nextNode=nextNode->next;
        }
        if(minNode!=current){
            int temp=current->data;
            current->data=minNode->data;
            minNode->data=temp;
        }
        display(head);
        current=current->next;
    }
}

int main(){
    struct Node* head=NULL;
    int n,value;
    printf("Enter size: ");
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        scanf("%d",&value);
        append(&head,value);
    }
    printf("Sorting process:\n");
    selectionSort(head);
    printf("Sorted list: ");
    display(head);
    return 0;
}
```

Sample Input & Output

```
Enter size: 4

2034

3420

34

20

Sorting process:
```

```
20 3420 34 2034

20 34 3420 2034

20 34 2034 3420

20 34 2034 3420

Sorted list: 20 34 2034 3420
```

Q5) Write a function to merge two sorted linked lists into one sorted linked list. Do not use arrays or extra data structures.

```c
#include <stdio.h>
#include <stdlib.h>

struct Node{
   int data;
   struct Node *next;
};

struct Node* createNode(int value){
   struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
   newNode->data=value;
   newNode->next=NULL;
   return newNode;
}

void append(struct Node** head,int value){
   struct Node* newNode=createNode(value);
   if(*head==NULL){
      *head=newNode;
      return;
   }
   struct Node* temp=*head;
   while(temp->next!=NULL){
      temp=temp->next;
   }
   temp->next=newNode;
}

void display(struct Node* head){
    struct Node* temp=head;
    while(temp!=NULL){
        printf("%d ",temp->data);
        temp=temp->next;
    }
    printf("\n");
}

struct Node* mergeLists(struct Node* a,struct Node* b){
    if(a==NULL)return b;
    if(b==NULL)return a;
    struct Node* result=NULL;
    if(a->data<=b->data){
        result=a;
        result->next=mergeLists(a->next,b);
    }else{
        result=b;
        result->next=mergeLists(a,b->next);
    }
    return result;
}

int main(){
    struct Node* list1=NULL;
    struct Node* list2=NULL;
    int n1,n2,val;
    printf("Enter size of first list: ");
    scanf("%d",&n1);
    for(int i=0;i<n1;i++){
        scanf("%d",&val);
        append(&list1,val);
    }
    printf("Enter size of second list: ");
    scanf("%d",&n2);
    for(int i=0;i<n2;i++){
        scanf("%d",&val);
        append(&list2,val);
    }
    printf("Merged sorted list: ");
    struct Node* merged=mergeLists(list1,list2);
    display(merged);
    return 0;
```

```
}
```

```
Sample Input & Output
Enter size of first list: 3
20
34
20334   34
Enter size of second list: 4
680
2
7
9
Merged sorted list: 20 34 680 2 7 9 2034
```

Q6) Implement a merge procedure using two queues (each containing sorted integers) and output a single sorted queue.

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

int queue1[MAX],queue2[MAX],mergedQueue[MAX];
int front1=0,rear1=-1,front2=0,rear2=-1,frontM=0,rearM=-1;

void enqueue(int queue[],int *rear,int val){
   queue[++(*rear)]=val;
}

int dequeue(int queue[],int *front){
   return queue[(*front)++];
}

int isEmpty(int front,int rear){
   return front>rear;
}
```

```
void display(int queue[],int front,int rear){
   for(int i=front;i<=rear;i++){
      printf("%d ",queue[i]);
   }
   printf("\n");
}

void mergeQueues(){
   while(!isEmpty(front1,rear1)&&!isEmpty(front2,rear2)){
      if(queue1[front1]<=queue2[front2]){
         enqueue(mergedQueue,&rearM,dequeue(queue1,&front1));
      }else{
         enqueue(mergedQueue,&rearM,dequeue(queue2,&front2));
      }
   }
   while(!isEmpty(front1,rear1)){
      enqueue(mergedQueue,&rearM,dequeue(queue1,&front1));
   }
   while(!isEmpty(front2,rear2)){
      enqueue(mergedQueue,&rearM,dequeue(queue2,&front2));
   }
}

int main(){
   int n1,n2,val;
   printf("Enter size of first queue: ");
   scanf("%d",&n1);
   for(int i=0;i<n1;i++){
      scanf("%d",&val);
      enqueue(queue1,&rear1,val);
   }
   printf("Enter size of second queue: ");
   scanf("%d",&n2);
   for(int i=0;i<n2;i++){
      scanf("%d",&val);
      enqueue(queue2,&rear2,val);
   }
   mergeQueues();
   printf("Merged sorted queue: ");
   display(mergedQueue,frontM,rearM);
   return 0;
}
```

```
Sample Input & Output
Enter size of first queue: 3
20
34
2034
Enter size of second queue: 2
3420
680
Merged sorted queue: 20 34 2034 3420 680
```

Q7) Write a hybrid program that divides an array into chunks, performs selection sort on each chunk, and then merges the chunks using your merge procedure.

```
#include <stdio.h>

void selectionSort(int arr[],int start,int end){
   int i,j,minIndex,temp;
   for(i=start;i<end-1;i++){
      minIndex=i;
      for(j=i+1;j<end;j++){
         if(arr[j]<arr[minIndex]){
            minIndex=j;
         }
      }
      temp=arr[i];
      arr[i]=arr[minIndex];
      arr[minIndex]=temp;
   }
}

void merge(int arr[],int temp[],int left,int mid,int right){
   int i=left,j=mid,k=left;
   while(i<mid && j<=right){
        if(arr[i]<=arr[j]){
            temp[k++]=arr[i++];
        }else{
            temp[k++]=arr[j++];
        }
    }
    while(i<mid){
        temp[k++]=arr[i++];
    }
    while(j<=right){
        temp[k++]=arr[j++];
    }
    for(i=left;i<=right;i++){
        arr[i]=temp[i];
    }
}

int main(){
    int n,chunkSize;
    printf("Enter size of array: ");
    scanf("%d",&n);
    int arr[n],temp[n];
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    printf("Enter chunk size: ");
    scanf("%d",&chunkSize);
    for(int i=0;i<n;i+=chunkSize){
        int end=i+chunkSize;
        if(end>n)end=n;
        selectionSort(arr,i,end);
    }
    for(int size=chunkSize;size<n;size*=2){
        for(int left=0;left<n-size;left+=2*size){
            int mid=left+size;
            int right=left+2*size-1;
            if(right>=n)right=n-1;
            merge(arr,temp,left,mid,right);
        }
    }
    printf("Final merged sorted array: ");
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
    printf("\n");
    return 0;
}
```

Sample Input & Output

```
Enter size of array: 4

34  2034

34

2034

20

Enter chunk size: 2

Final merged sorted array: 20 34 2034 2034
```

Q8) Implement a stack-based simulation of the merge process where push and pop operations are used instead of recursion.

```
#include <stdio.h>
#include <stdlib.h>

struct Item{int left;int right;int visited;};

void merge(int arr[],int temp[],int left,int mid,int right){
    int i=left,j=mid+1,k=left;
    while(i<=mid && j<=right){
        if(arr[i]<=arr[j]) temp[k++]=arr[i++];
        else temp[k++]=arr[j++];
    }
    while(i<=mid) temp[k++]=arr[i++];
    while(j<=right) temp[k++]=arr[j++];
    for(i=left;i<=right;i++) arr[i]=temp[i];
}

int main(){
    int n;
    printf("Enter number of elements: ");
    scanf("%d",&n);
    if(n<=0){
        printf("Invalid size\n");
        return 0;
    }

    int *arr=malloc(n*sizeof(int));
    int *temp=malloc(n*sizeof(int));

    printf("Enter %d elements: ",n);
    for(int i=0;i<n;i++) scanf("%d",&arr[i]);

    int stackSize=2*n+5;
    struct Item *stack=malloc(stackSize*sizeof(struct Item));
    int top=-1;
    stack[++top]=(struct Item){0,n-1,0};

    while(top>=0){
        struct Item cur=stack[top--];
        int L=cur.left;
        int R=cur.right;
        if(L>=R) continue;
        if(cur.visited==0){
            stack[++top]=(struct Item){L,R,1};
            int mid=(L+R)/2;
            stack[++top]=(struct Item){mid+1,R,0};
            stack[++top]=(struct Item){L,mid,0};
        }else{
            int mid=(L+R)/2;
            merge(arr,temp,L,mid,R);
        }
    }

    printf("Sorted array: ");
    for(int i=0;i<n;i++) printf("%d ",arr[i]);
    printf("\n");

    free(arr);
    free(temp);
    free(stack);
    return 0;
}
```

Sample Input & Output

```
Enter number of elements: 4

Enter 4 elements: 3420

20

34

20334   4 34

Sorted array: 20 34 2034 3420
```