DSA Lab08

23K2001

M.Muzammil Siddiqui

BCS-3J

Q1:

//23K2001 - Muzammil

#include<iostream>

using namespace std;

class student{

    public:

        string name;

        int score;

        student(){}

        student(string n, int s):name(n),score(s){}

        void getData(){

            cout<<"Name: "<<name<<"\t"<<"Score: "<<score<<endl;

        }

        void operator =(const student s){

            this->name = s.name;

            this->score = s.score;

        }

};

class node{

    public:

        student std;

        node\* next;

        node(student s) : std(s),next(nullptr){}

};

class singleList{

    private:

        node\* head;

        node\* tail;

    public:

        singleList(){

            head = nullptr;

            tail = nullptr;

        }

        void display(){

            node\* temp = head;

            while(temp!=nullptr)

            {

                temp->std.getData();

                temp=temp->next;

            }

            cout<<endl;

        }

        void insertAtEnd(student s){

            node\* temp = head;

            node\* n = new node(s);

            if(head == NULL){

                head = n;

                tail = n;

            }

            else{

                tail->next = n;

                tail = n;

            }

        }

        void deleteNode(string n,int s){

            node\* before = nullptr;

            node\* temp = head;

            if(temp != nullptr && temp->std.name == n && temp->std.score == s) {

                head = temp->next;

                delete temp;

                if(head == nullptr)

                    tail = nullptr;

                return;

            }

            while(temp!=nullptr){

                if(temp->std.name==n && temp->std.score==s){

                    before->next = (temp->next);

                    delete temp;

                    return;

                }

                before = temp;

                temp = temp->next;

            }

            cout<<"Record was not found!"<<endl;

        }

        student getMax(){

            node\* temp = head;

            student m = head->std;

            while(temp != nullptr){

                if (temp->std.score > m.score)

                    m = temp->std;

                temp = temp->next;

            }

            return m;

        }

        int numOfnodes(){

            node\* temp = head;

            int c=0;

            while(temp!=nullptr){

                c++;

                temp = temp->next;

            }

            return c;

        }

        void countingSort(int place){

            int n = numOfnodes();

            student\* output = new student[n];

            int count[10] = {0};

            node\* temp = head;

            while (temp != nullptr){

                count[(temp->std.score / place) % 10]++;

                temp = temp->next;

            }

            for (int i = 1; i < 10; i++)

                count[i] += count[i - 1];

            temp = head;

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

                output[count[(temp->std.score / place) % 10] - 1] = temp->std;

                count[(temp->std.score / place) % 10]--;

                temp = temp->next;

            }

            temp = head;

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

                temp->std = output[i];

                temp = temp->next;

            }

            delete[] output;

        }

        void radixSort() {

            node\* maxNode = head;

            node\* temp = head;

            while (temp != nullptr) {

                if (temp->std.score > maxNode->std.score)

                    maxNode = temp;

                temp = temp->next;

            }

            int maxScore = maxNode->std.score;

            for (int place = 1; maxScore / place > 0; place \*= 10)

                countingSort(place);

        }

        node\* getMiddle(node\* start, node\* end){

            if(start == nullptr)

                return nullptr;

            node\* slow = start;

            node\* fast = start;

            while (fast != end && fast->next != end) {

                slow = slow->next;

                fast = fast->next->next;

            }

            return slow;

        }

        node\* binarySearch(string n,int s){

            node\* left = head;

            node\* right = nullptr;

            while (left != right) {

                node\* mid = getMiddle(left, right);

                if (mid == nullptr) return nullptr;

                if (mid->std.name==n && mid->std.score==s)

                    return mid;

                else if (mid->std.score < s)

                    left = mid->next;

                else

                    right = mid;

            }

            return nullptr;

        }

};

int main(){

    student ayaan("Ayaan", 90);

    student zameer("Zameer", 60);

    student sara("Sara", 70);

    student sohail("Sohail", 30);

    student ahmed("Ahmed", 20);

    singleList flex;

    flex.insertAtEnd(ayaan);

    flex.insertAtEnd(zameer);

    flex.insertAtEnd(sara);

    flex.insertAtEnd(sohail);

    flex.insertAtEnd(ahmed);

    cout<<"Your list: "<<endl;

    flex.display();

    cout<<"Applying Radix Sort!"<<endl<<endl;

    flex.radixSort();

    cout<<"Your list: "<<endl;

    flex.display();

    string n;

    int s;

    cout<<"Enter name & score to search:"<<endl;

    cin>>n>>s;

    node\* src = flex.binarySearch(n,s);

    if(src){

        cout<<"Record with Name: "<<src->std.name<<", Score: "<<src->std.score<<" found!"<<endl;

        flex.deleteNode(src->std.name,src->std.score);

        cout<<"Successfully deleted."<<endl<<endl;

    }

    else

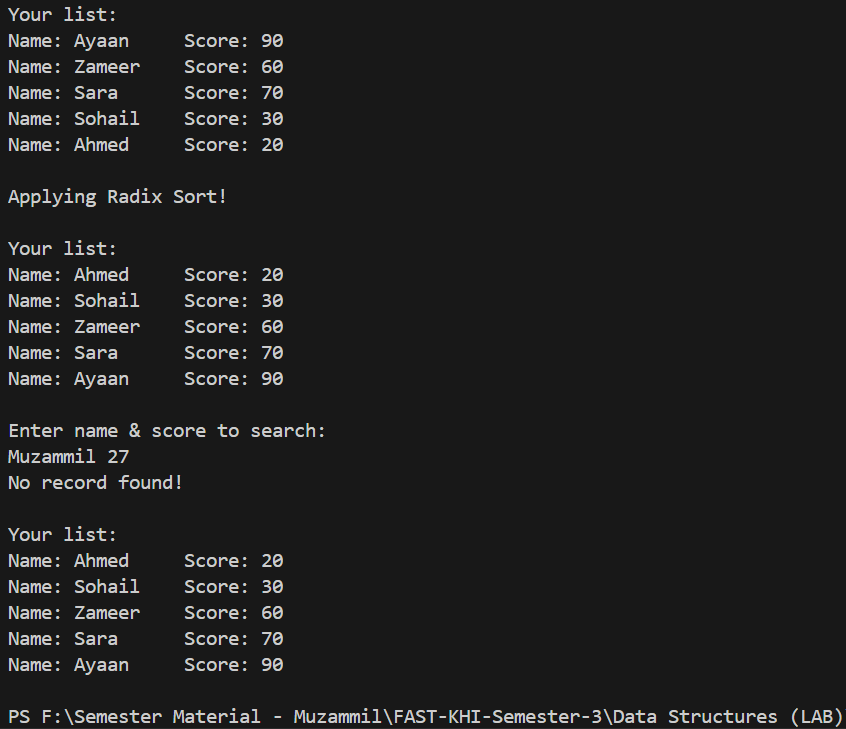
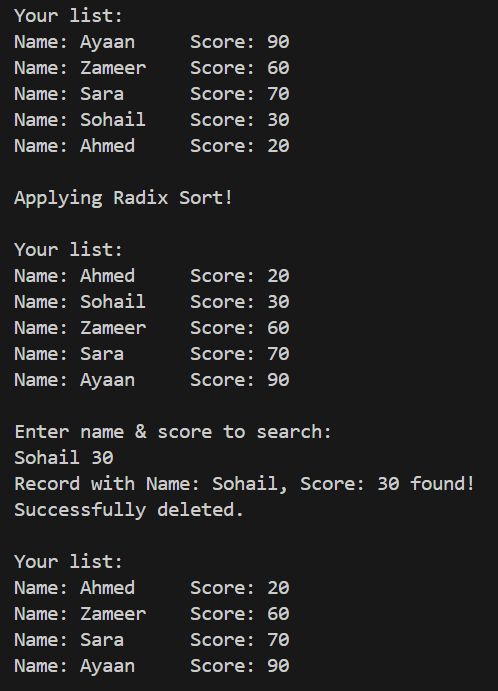
        cout<<"No record found!"<<endl<<endl;

    cout<<"Your list:"<<endl;

    flex.display();

    return 0;

}



Q2:

//23K2001 - Muzammil

#include<iostream>

using namespace std;

int getMax(int a[], int n) {

   int max = a[0];

   for(int i = 1; i<n; i++) {

      if(a[i] > max)

        max = a[i];

   }

   return max;

}

void countingSortAsc(int a[], int n, int place){

  int output[n + 1];

  int count[10] = {0};

  for (int i = 0; i < n; i++)

    count[(a[i] / place) % 10]++;

  for (int i = 1; i < 10; i++)

    count[i] += count[i - 1];

  for (int i = n - 1; i >= 0; i--) {

    output[count[(a[i] / place) % 10] - 1] = a[i];

    count[(a[i] / place) % 10]--;

  }

  for (int i = 0; i < n; i++)

    a[i] = output[i];

}

void countingSortDesc(int a[], int n, int place){

    int output[n + 1];

    int count[10] = {0};

    for (int i = 0; i < n; i++)

        count[(a[i] / place) % 10]++;

    for (int i = 8; i >= 0; i--)

        count[i] += count[i + 1];

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

        output[count[(a[i] / place) % 10] - 1] = a[i];

        count[(a[i] / place) % 10]--;

    }

    for (int i = 0; i < n; i++)

        a[i] = output[i];

}

void radixSortAscend(int a[], int n){

  int max = getMax(a, n);

  for (int place = 1; max / place > 0; place \*= 10)

    countingSortAsc(a, n, place);

}

void radixSortDescend(int a[], int n){

    int max = getMax(a, n);

    for (int place = 1; max / place > 0; place \*= 10)

        countingSortDesc(a, n, place);

}

void display(int a[], int n) {

    for(int i = 0; i < n; ++i)

        cout<<a[i]<<" ";

    cout<<endl<<endl;

}

int main(){

    int a[] = {36,987, 654, 2, 20, 99, 456, 957, 555, 420, 66, 3};

    int b[] = {36,987, 654, 2, 20, 99, 456, 957, 555, 420, 66, 3};

    int n = sizeof(a) / sizeof(a[0]);

    cout<<"Before sorting array elements are - \n";

    display(a,n);

    radixSortAscend(a, n);

    cout<<"After applying Radix sort (ascending), the array elements are - \n";

    display(a, n);

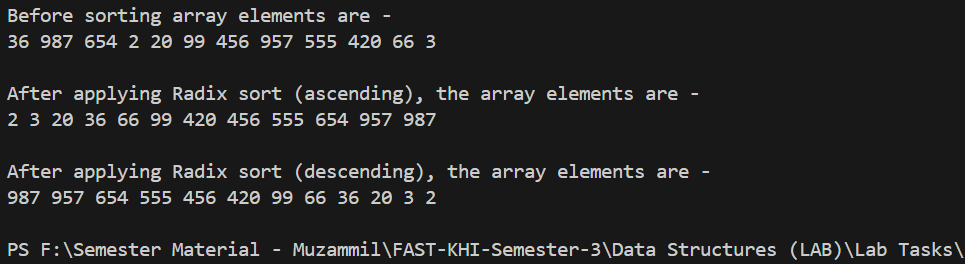
    radixSortDescend(b, n);

    cout<<"After applying Radix sort (descending), the array elements are - \n";

    display(b, n);

    return 0;

}



Q3:

//23K2001 - Muzammil

#include<iostream>

using namespace std;

class runner{

    public:

        string name;

        float time;

        runner(){}

        runner(string n,float t):name(n),time(t){}

};

void merge(runner \*a, int beg, int mid, int end){

    int i, j, k;

    int n1 = mid - beg + 1;

    int n2 = end - mid;

    runner LeftArray[n1], RightArray[n2];

    for (int i = 0; i < n1; i++){

    LeftArray[i].time = a[beg + i].time;

    LeftArray[i].name = a[beg + i].name;

    }

    for (int j = 0; j < n2; j++){

    RightArray[j].time = a[mid + 1 + j].time;

    RightArray[j].name = a[mid + 1 + j].name;

    }

    i = 0;

    j = 0;

    k = beg;

    while (i < n1 && j < n2)

    {

        if(LeftArray[i].time <= RightArray[j].time)

        {

            a[k].name = LeftArray[i].name;

            a[k].time = LeftArray[i].time;

            i++;

        }

        else

        {

            a[k].name = RightArray[j].name;

            a[k].time = RightArray[j].time;

            j++;

        }

        k++;

    }

    while (i<n1)

    {

        a[k].time = LeftArray[i].time;

        a[k].name = LeftArray[i].name;

        i++;

        k++;

    }

    while (j<n2)

    {

        a[k].time = RightArray[j].time;

        a[k].name = RightArray[j].name;

        j++;

        k++;

    }

}

void mergeSort(runner \*a, int beg, int end){

    if (beg < end)

    {

        int mid = (beg + end) / 2;

        mergeSort(a, beg, mid);

        mergeSort(a, mid + 1, end);

        merge(a, beg, mid, end);

    }

}

void display(runner \*a, int n){

    for(int i = 0; i < n; ++i)

        cout<<i+1<<". "<<a[i].name<<"\t"<<a[i].time<<" (sec)"<<endl;

    cout<<endl;

}

int main(){

    int n;

    cout<<"How many runners: ";

    cin>>n;

    runner \*runners = new runner[n];

    for(int i=0;i<n;i++){

        cout<<"Input name & finish time (sec) for runner#"<<i+1;

        cout<<": ";

        cin>>runners[i].name>>runners[i].time;

    }

    cout<<endl;

    display(runners,n);

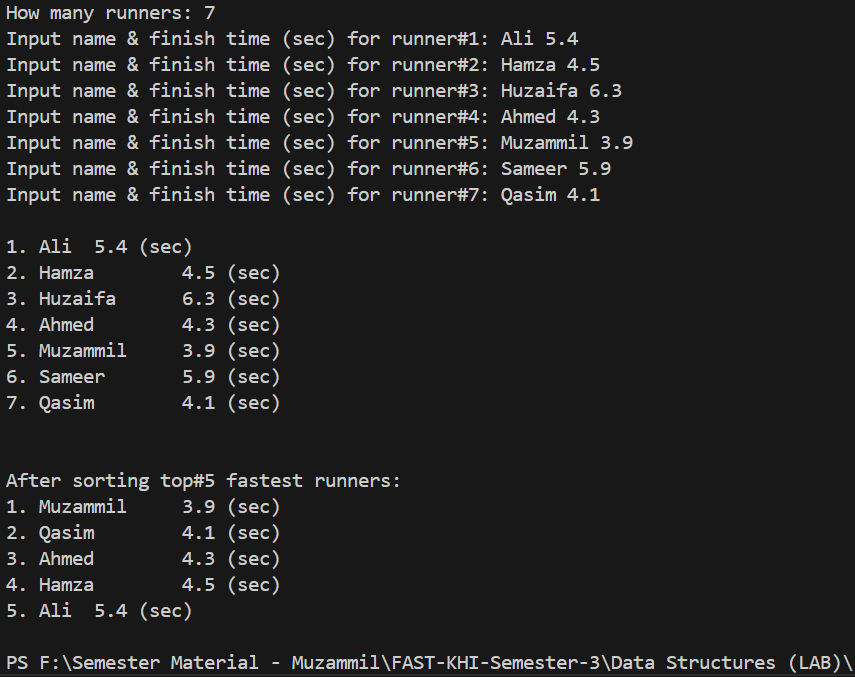
    cout<<endl<<"After sorting top#5 fastest runners:"<<endl;

    mergeSort(runners,0,n-1);

    display(runners,5);

    return 0;

}



Q4:

//23K2001 - Muzammil

#include<iostream>

using namespace std;

class product{

    public:

        string name,desc;

        float price;

        bool availability;

        product(){}

        product(string n,float p,string d,bool a): name(n),desc(d),price(p),availability(a){}

};

int partition(product a[], int start, int end){

    float pivot = a[end].price;

    int i = start - 1;

    for(int j = start; j < end; j++) {

        if (a[j].price <= pivot) {

            i++;

            product t = a[i];

            a[i] = a[j];

            a[j] = t;

        }

    }

    product t = a[i + 1];

    a[i + 1] = a[end];

    a[end] = t;

    return i + 1;

}

void quickSort(product a[], int start, int end){

    if (start < end)

    {

        int p = partition(a, start, end);

        quickSort(a, start, p - 1);

        quickSort(a, p + 1, end);

    }

}

void display(product \*p,int n){

    for(int i=0;i<n;i++)

        cout<<p[i].name<<" - "<<p[i].price<<endl;

    cout<<endl;

}

int main(){

    product p1 = {"Product 1",10.99,"This is product 1.",true};

    product p2 = {"Product 2",5.99,"This is product 2.",false};

    product p3 = {"Product 3",2.99,"This is product 3.",true};

    product products[] = {p1,p2,p3};

    cout<<"Original entries before sorting:"<<endl;

    display(products,3);

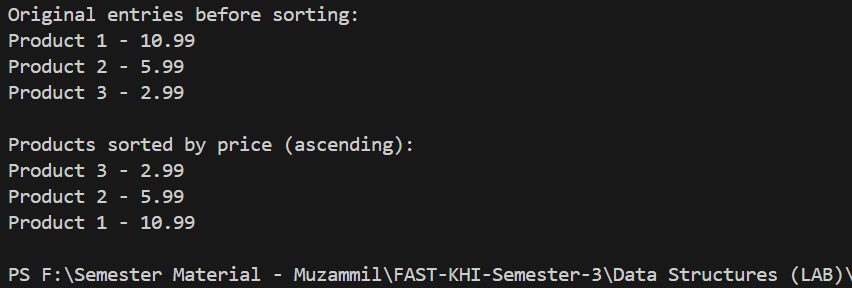
    cout<<"Products sorted by price (ascending):"<<endl;

    quickSort(products,0,2);

    display(products,3);

    return 0;

}



Q5:

//23K2001 - Muzammil

#include<iostream>

using namespace std;

class node{

    public:

        int data;

        node\* next;

        node() : next(nullptr) {}

        node(int d) : data(d), next(nullptr){}

        void setNext(node\* n){ next = n; }

        void setData(int d){ data = d; }

};

class singleList{

    public:

        node\* head;

        node\* tail;

        singleList() : head(nullptr), tail(nullptr){}

        void display(){

            if(!head){

                cout<<"List is empty!"<<endl;

                return;

            }

            cout<<"Elements in the list are:"<<endl;

            node\* temp = head;

            while(temp){

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

                temp = temp->next;

            }

            cout<<endl;

        }

        void insertAtEnd(int d){

            node\* n = new node(d);

            if(!head){

                head = n;

                tail = n;

            }

            else{

                tail->next = n;

                tail = n;

            }

        }

        void deleteNode(int d){

            if(!head){

                cout<<"List is empty!"<<endl;

                return;

            }

            node\* before = nullptr;

            node\* temp = head;

            while(temp->data!=d){

                before = temp;

                temp = temp->next;

            }

            if(!temp){

                cout<<"Value "<<d<<" not found in the list."<<endl;

                return;

            }

            if(temp == head)

                head = head->next;

            else

                before->next = temp->next;

            delete temp;

            cout<<"node with value "<<d<<" deleted.\n";

        }

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

            if (a == b) return;

            int temp = a->data;

            a->data = b->data;

            b->data = temp;

        }

        node\* partition(node\* start, node\* end){

            int pivot = end->data;

            node\* i = start;

            node\* j = start;

            while (j != end) {

                if (j->data < pivot) {

                    swap(i, j);

                    i = i->next;

                }

                j = j->next;

            }

            swap(i, end);

            return i;

        }

        void quickSort(node\* start, node\* end) {

            if(head == nullptr || head->next == nullptr){

                cout<<"Quick sort cannot be applied on this list!"<<endl;

                return;

            }

            if(start != end){

                node\* pivot = partition(start, end);

                node\* temp = start;

                while(temp && temp->next != pivot)

                    temp = temp->next;

                if (temp != nullptr)

                    quickSort(start, temp);

                quickSort(pivot->next, end);

            }

        }

};

int main(){

    singleList flex1,flex2,flex3;

    cout<<"Empty list (flex1):"<<endl;

    cout<<"Applying quick sort.."<<endl<<endl;

    flex1.quickSort(flex1.head,flex1.tail);

    flex2.insertAtEnd(5);

    cout<<endl<<"List with single node (flex2):"<<endl;

    flex2.display();

    cout<<"Applying quick sort.."<<endl<<endl;

    flex2.quickSort(flex2.head,flex2.tail);

    flex3.insertAtEnd(10);

    flex3.insertAtEnd(7);

    flex3.insertAtEnd(8);

    flex3.insertAtEnd(9);

    flex3.insertAtEnd(1);

    flex3.insertAtEnd(5);

    flex3.insertAtEnd(3);

    cout<<endl<<"Sample list (flex3):"<<endl;

    flex3.display();

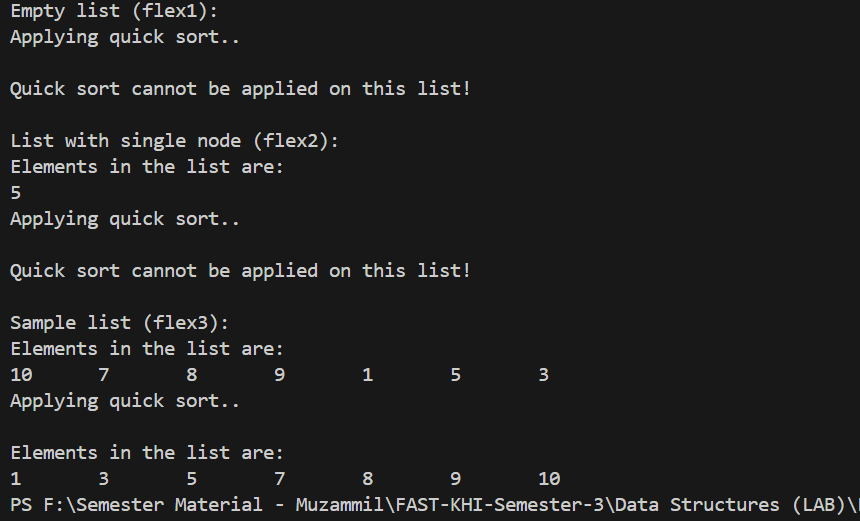
    cout<<"Applying quick sort.."<<endl<<endl;

    flex3.quickSort(flex3.head,flex3.tail);

    flex3.display();

    return 0;

}



Q6:

//23K2001 - Muzammil

#include<iostream>

using namespace std;

int getMax(int a[], int n) {

   int max = a[0];

   for(int i = 1; i<n; i++) {

      if(a[i] > max)

         max = a[i];

   }

   return max;

}

void countingSort(int a[], int n, int place){

  int output[n + 1];

  int count[10] = {0};

  for (int i = 0; i < n; i++)

    count[(a[i] / place) % 10]++;

  for (int i = 1; i < 10; i++)

    count[i] += count[i - 1];

  for (int i = n - 1; i >= 0; i--){

    output[count[(a[i] / place) % 10] - 1] = a[i];

    count[(a[i] / place) % 10]--;

  }

  for (int i = 0; i < n; i++)

    a[i] = output[i];

}

void radixsort(int a[], int n){

  int max = getMax(a, n);

  for (int place = 1; max / place > 0; place \*= 10)

    countingSort(a, n, place);

}

void merge(int a[], int beg, int mid, int end){

    int i, j, k;

    int n1 = mid - beg + 1;

    int n2 = end - mid;

    int LeftArray[n1], RightArray[n2];

    for (int i = 0; i < n1; i++)

        LeftArray[i] = a[beg + i];

    for (int j = 0; j < n2; j++)

        RightArray[j] = a[mid + 1 + j];

    i = 0;

    j = 0;

    k = beg;

    while (i < n1 && j < n2){

        if(LeftArray[i] <= RightArray[j]){

            a[k] = LeftArray[i];

            i++;

        }

        else{

            a[k] = RightArray[j];

            j++;

        }

        k++;

    }

    while (i<n1){

        a[k] = LeftArray[i];

        i++;

        k++;

    }

    while (j<n2){

        a[k] = RightArray[j];

        j++;

        k++;

    }

}

void mergeSort(int a[], int beg, int end){

    if (beg < end){

        int mid = (beg + end) / 2;

        mergeSort(a, beg, mid);

        mergeSort(a, mid + 1, end);

        merge(a, beg, mid, end);

    }

}

void combineArr(int \*&dest,int arr1[],int arr2[],int n1,int n2,int &destSize){

    dest = new int[n1+n2];

    destSize = n1+n2;

    int i;

    for(i=0;i<n1;i++)

        dest[i] = arr1[i];

    for(int j=0;j<n2;j++)

        dest[i+j] = arr2[j];

}

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

    for(int i=0;i<n;i++)

        cout<<arr[i]<<"\t";

    cout<<endl;

}

int main(){

    int arr1[10] = {1,10,2,9,3,8,4,7,5,6};

    int arr2[10] = {0,11,20,12,19,13,18,17,14,16};

    cout<<"Array#1:"<<endl;

    display(arr1,10);

    cout<<endl<<"Array2:"<<endl;

    display(arr2,10);

    int newSize;

    int \*combined;

    combineArr(combined,arr1,arr2,10,10,newSize);

    cout<<endl<<"Combined Array:"<<endl;

    display(combined,newSize);

    int rSorted[newSize];

    int mSorted[newSize];

    for(int i=0;i<newSize;i++){

        rSorted[i] = combined[i];

        mSorted[i] = combined[i];

    }

    cout<<endl<<"After sorting with Radix Sort:"<<endl;

    radixsort(rSorted,newSize);

    display(rSorted,newSize);

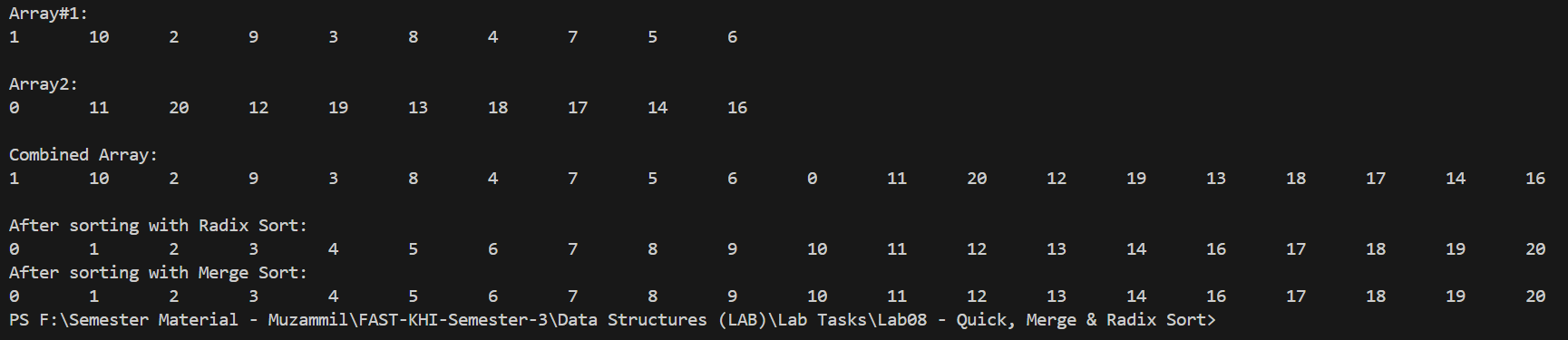
    cout<<"After sorting with Merge Sort:"<<endl;

    mergeSort(mSorted,0,newSize-1);

    display(mSorted,newSize);

    return 0;

}



Q7:

//23K2001 - Muzammil

#include<iostream>

#include<cstdlib>

#include<time.h>

using namespace std;

int random(int i,int j){

    return (i + rand()%(j-i+1));

}

void swap(int &a,int &b){

    int temp = a;

    a = b;

    b = temp;

}

int count = 0;

int partition(int a[], int start, int end, int pivot){

    int i = start;

    int j = end;

    while (i < j){

        while (a[i] <= a[pivot] && i < end){

            i++;

            count++;

        }

        while (a[j] > a[pivot] && j > start){

            j--;

            count++;

        }

        if (i < j)

            swap(a[i], a[j]);

    }

    swap(a[start], a[j]);

    return j;

}

void qSort\_FirstElement(int a[],int start,int end){

    if (start < end)

    {

        int pivot = start;

        int p = partition(a, start, end, pivot);

        qSort\_FirstElement(a, start, p - 1);

        qSort\_FirstElement(a, p + 1, end);

    }

}

void qSort\_RandomElement(int a[],int start,int end){

    if (start < end)

    {

        int r = random(start,end);

        swap(a[start],a[r]);

        int pivot = start;

        int p = partition(a, start, end, pivot);

        qSort\_RandomElement(a, start, p - 1);

        qSort\_RandomElement(a, p + 1, end);

    }

}

void qSort\_MidElement(int a[],int start,int end){

    if (start < end)

    {

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

        swap(a[start],a[mid]);

        int pivot = start;

        int p = partition(a, start, end, pivot);

        qSort\_MidElement(a, start, p - 1);

        qSort\_MidElement(a, p + 1, end);

    }

}

void qSort\_Median(int a[],int start,int end){

    if (start < end)

    {

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

        if ((a[start] <= a[mid] && a[mid] <= a[end]) || (a[end] <= a[mid] && a[mid] <= a[start]))

            swap(a[start], a[mid]);

        else if ((a[mid] <= a[start] && a[start] <= a[end]) || (a[end] <= a[start] && a[start] <= a[mid]))

            swap(a[start], a[start]);

        else

            swap(a[start], a[end]);

        int pivot = start;

        int p = partition(a, start, end, pivot);

        qSort\_Median(a, start, p - 1);

        qSort\_Median(a, p + 1, end);

    }

}

void display(int \*p,int n){

    for(int i=0;i<n;i++)

        cout<<p[i]<<"\t";

    cout<<endl;

}

int main(){

    srand(time(0));

    int flex1[] = {5,1,4,2,3};

    cout<<"QuickSort (pivot: first element) on Randomized Array:"<<endl;

    qSort\_FirstElement(flex1,0,4);

    display(flex1,5);

    cout<<"\t\tComparisons made: "<<count<<endl<<endl;

    count = 0;

    int flex2[] = {5,1,4,2,3};

    cout<<"QuickSort (pivot: middle element) on Randomized Array:"<<endl;

    qSort\_MidElement(flex2,0,4);

    display(flex2,5);

    cout<<"\t\tComparisons made: "<<count<<endl<<endl;

    count = 0;

    int flex3[] = {5,1,4,2,3};

    cout<<"QuickSort (pivot: median element) on Randomized Array:"<<endl;

    qSort\_Median(flex3,0,4);

    display(flex3,5);

    cout<<"\t\tComparisons made: "<<count<<endl<<endl;

    count = 0;

    cout<<"-------------------------------------------------"<<endl;

//-----------------------------------------------------------------------------------------------

    int flex4[] = {1,2,3,4,5};

    cout<<"QuickSort (pivot: first element) on Sorted Array:"<<endl;

    qSort\_FirstElement(flex4,0,4);

    display(flex4,5);

    cout<<"\t\tComparisons made: "<<count<<endl<<endl;

    count = 0;

    int flex5[] = {1,2,3,4,5};

    cout<<"QuickSort (pivot: middle element) on Sorted Array:"<<endl;

    qSort\_MidElement(flex5,0,4);

    display(flex5,5);

    cout<<"\t\tComparisons made: "<<count<<endl<<endl;

    count = 0;

    int flex6[] = {1,2,3,4,5};

    cout<<"QuickSort (pivot: median element) on Sorted Array:"<<endl;

    qSort\_Median(flex6,0,4);

    display(flex6,5);

    cout<<"\t\tComparisons made: "<<count<<endl<<endl;

    count = 0;

    cout<<"-------------------------------------------------"<<endl;

//-----------------------------------------------------------------------------------------------

    int flex7[] = {5,4,3,2,1};

    cout<<"QuickSort (pivot: first element) on Reverse-Sorted Array:"<<endl;

    qSort\_FirstElement(flex7,0,4);

    display(flex7,5);

    cout<<"\t\tComparisons made: "<<count<<endl<<endl;

    count = 0;

    int flex8[] = {5,4,3,2,1};

    cout<<"QuickSort (pivot: middle element) on Reverse-Sorted Array:"<<endl;

    qSort\_MidElement(flex8,0,4);

    display(flex8,5);

    cout<<"\t\tComparisons made: "<<count<<endl<<endl;

    count = 0;

    int flex9[] = {5,4,3,2,1};

    cout<<"QuickSort (pivot: median element) on Reverse-Sorted Array:"<<endl;

    qSort\_Median(flex9,0,4);

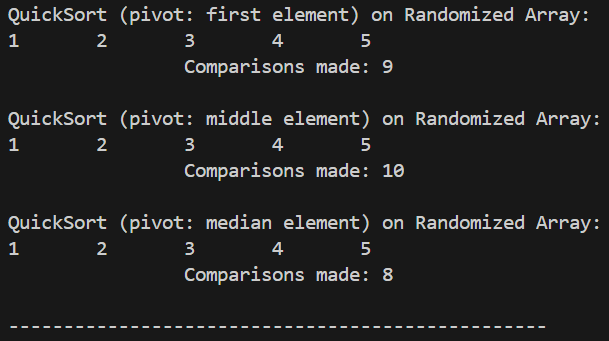
    display(flex9,5);

    cout<<"\t\tComparisons made: "<<count<<endl<<endl;

    count = 0;

    return 0;

}



A computer screen shot of a black screen

Description automatically generatedA screen shot of a computer

Description automatically generated