Jake Rounds
Section 519
UIN: 730000964

Notes

Students: Cory Overgaard, Haden Johnson, Jonathan Hicks.
Resources Used: GeeksforGeeks.org, Peer teachers in PETR 127.

Number 6 Answer

I ran 2a, 2b, 4a, and 4b multiple times each. The averages of the 6 times are listed below.

2a: 86.5
2b: 338.83
4a: 819.5
4b: 228.5

Given these values listed above, I found that algorithms with the linked list take longer on average. This is due to the need of traversing through the entire linked list whereas vectors and pointers have more flexibility in terms of size, indexes.

Instructions for 9

For question 9 I implemented the hexadecimal sequence as a vector of strings. Please replace my strings for yours on line 181 of 9.cpp. Additionally, I implemented the operands as an array, please place your initial values for a-e in the array on line 182 or 9.cpp. Thank you

Final Pseudocode

1a

```
Class node{
    public :
        constructor
        Data
        Next
        Prev
}

Class cubelist{
    public:
        constructor
        destructor
        append _end(int a)
```

```
                print()
}

Void append_end(int a){
        if(first element)
                head.next = new_node
                tail.prev = new_node
        else
                new_node1->prev->next = new_node1
                tail->prev = new_node1
}

Void print(){
        while(new_node->next != NULL)
                Cout << new_node1->data
                new_node1 = new_node1->next
}

Int main(){
        //Pointer
        Int *pointer = (int *) malloc(300*sizeof(int))

        Int even_index = 0;
        for(i = 150, i < 450; i+=2)
                Pointer[even_index] = i
                even_index++

        Int odd_index = 0;
        for(i = 449, i > 149; i-=2)
                pointer[odd_index] = i
                odd_index++

        //Print
        for(i = 0, i < 300, i++)
                cout << pointer[i] << " "

        //Vector
        vector<int> vect;

        for(i = 150, i < 450, i+= 2)
                vect.push_back(i)

        for(i = 449, i > 149, i -=2)
                vect.push_back(i)
```

```
        //Print
        for(i = 0, i < 300, i++)
                cout << vect.at(i) << " "

        //Linked list
        Cubelist list1

        for(i = 150, i < 450; i+=2)
                list1.append_end(i)


        for(i = 449, i > 149; i-=2)
                list1.append_end(i)

        list1.print()

}
```

2a

```
Void insertion_sort(int* pointer){
        j = 0;
        temp = 0;
        for(i = 0, i < 300, i++)
                temp = pointer[i]
                J = i-1

                while(j >= 0 && pointer[j] > temp)
                        pointer[j+1] = pointer[j]
                        J -= 1

                pointer[j+1] = temp
}

Int main(){
        //Initialize pointer with proper values (see 1a)

        insertion_sort(pointer)

        //Print
```

```
        for(i = 0, i < 300, i++)
                cout << pointer[i] << " "
}
```

2b

```
Void sort(int* pointer){
        For(i = 0, i < 300, i++)
                for(j = 1, j < 1, j++)
                        if (pointer[j] > pointer[j-1] && pointer[i] < pointer[j])
                                Swap pointer[j] and pointer[i]

}

Int main(){
        //Initialize pointer with proper values (see 1a)

        sort(pointer)

        //Print
        for(i = 0, i < 300, i++)
                cout << pointer[i] << " "
}
```

3a

```
int  find_max(vector<int> vect, int start){
        max = 0;
        for(i = start, i < 300, i++)
                if(vect.at(i) > max)
                        max = vect.at(i)

}

Int main(){
        //Initialize vector correctly, see 1a

        //sort
        for(i = 0, i < 299, i++)
                x = find_max(vect, i)
                vect.insert(vect.begin(), x)

                for(int i = 300; i > 0, i--)
                        if(vect[i] == x)
                                for(j = i, j < 300, j++)
                                        vect[j] = vect[j+1]
```

```
        vect.insert(vect.begin(), 150)
        vect.erase(vect.begin()+299, vect.begin()+299)

        //Print
        for(i = 0, i < 300, i++)
                cout << vect.at(i) << " "
}
```

3b

```
vector<int> sort(vector<int vect>){
        temp = 0
        sorted = false
        while(sorted == false)
                sorted = true
                for(i = 0, i < 299, i++)
                        swap vect[i] and vect[i+1]
                        sorted = false

        return vect
}

Int main(){
        //Initialize vector correctly, see 1a

        vector<int> print_vect

        print_vect = sort(vect)

        //Print
        for(i = 0, i < 300, i++)
                cout << print_vect.at(i) << " "
}
```

4a

```
//Assume linked list is properly initialized, see 1a

Void swap(node2* a, node2* b){
        node2* temp = new node2(b->prev, b->next, b->data)

        b->next = a
        b->prev = a->prev
        a->prev->next = b
```

```
        a->prev = b
        a->next = temp->next
        temp->next->prev = a
        delete(temp)
}

Void insertion_sort(){
        node2* new_node1 = &head
        node2* new_node2 = &head

        temp = 0
        new_node1 = new_node1->next
        while(new_node1->next != NULL)
                temp = new_node1->data
                new_node2 = new_node1->prev

                while(new_node2 != NULL && new_node2->data > temp)
                        swap new_node2 and new_node2->next
                        new_node2 = new_node2->prev->prev


                new_node2 ->next->data = temp
                new_node1 = new_node1->next
}

Int main(){
        Cubelist list1

        //fill list with values, see 1a

        list1.insertion_sort()

        list1.print()
}
```

4b

```
//Assume linked list is properly initialized, see 1a

Void sort(){
        node2* new_node1 = &head
        node2* new_node2 = &head
```

```
        while(new_node1->next != NULL)
                New_node2 = new_node1->next
                while(new_node2->next != NULL)
                        if(new_node2->data < new_node1->data)
                                Swap new_node2 and new_node1

                        New_node2 = new_node2->next

                New_node1 = new_node1->next

}

Int main(){
        Cubelist list1

        //fill list with values, see 1a

        list1.sort()

        list1.print()
}
```

7a

```
//Assume vector, pointer and Linked list are all initialized properly and sorted, see 1a

void search_linked(){
        node2* search_node = &head

        while(search_node->next != NULL)
                if(search_node->data == 279)
                        Cout node address
}


Int main(){
        //Pointer
        for(i = 0, i < 300, i++)
                if(pointer[i] == 279)
                        Cout address and index

        //vector
        for(i = 0, i < 300, i++)
                if(vect[i] == 279)
```

```
                    Cout address and index

        list1.search_linked()
}
```

8a

```
//Assume linked list, pointer, and vector are properly initialized as well as sorted, see 1a

Void search_linked(int val){

}

void pointer_binary(int* point, int left1, int right1, int val1){
        middle = (left1+right1)/2
        if(point[middle] == val1)
                cout address and index
        else if(point[middle] != val && middle == 1)
                cout not found
        else
                if(val1 > point[middle])
                        Left1 = middle
                        pointer_binary(point, left1, right1, val1)
                else
                        Right1 = middle
                        pointer_binary(point, left1, right1,val1)
}

void vector_binary(vector<int> vec, int left2, int right2, int val2){
        middle = (left2+right2)/2
        if(vec[middle] == val2)
                cout address and index
        else if(vec[middle] != val && middle == 1)
                cout not found
        else
                if(val1 > vec[middle])
                        Left2 = middle
                        vector_binary(vec, left2, right2, val2)
                else
                        Right2 = middle
                        vector_binary(vec, left2, right2,val2)
}

node2* middle(node2* start, node2* end){
        node2* current = start
        node2* forward = start
```

```
        while(forward->next != end && forward->next->next != end)
                current = current->next
                forward = forward->next->next
        return current
}

Void search_linked(int val){
        node2* new_node1 = &head
        node2* new_node2 = &tail

        while(new_node1 != new_node2)
                node2* mid = middle(new_node1, new_node2)

                If(mid->data == val)
                        cout << &mid <<endl
                        return;
                else
                        if(mid->data > val)
                                new_node2 = mid
                        else
                                new_node1 = mid->next
}

Int main(){
        //couts are in the functions

        list1.search_linked(val)

}
```