

|  |  |  |  |
| --- | --- | --- | --- |
| Course Code | : | SOF105 | |
| Course Name | : | Data Structure | |
| Lecturer | : | Dr. Saif Kifah | |
| Academic Session | : | 2020/09 | |
| Assessment Title | : | Assignment 2 | |
| Submission Due Date | : | 9/1/2021 | |
| Prepared by |  |  | |
| : | Student ID | Student Name |
|  | DMT2002036 | Yim Jing Xiang |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| Date Received | : |  | |
|  |  |  | |
| Feedback from Lecturer:  Mark: | | | |

**Own Work Declaration**

I/We hereby understand my/our work would be checked for plagiarism or other misconduct, and the softcopy would be saved for future comparison(s).

I/We hereby confirm that all the references or sources of citations have been correctly listed or presented and I/we clearly understand the serious consequence caused by any intentional or unintentional misconduct.

This work is not made on any work of other students (past or present), and it has not been submitted to any other courses or institutions before.



Signature: Yim Jing Xiang

Date: 3/1/2021

**1. Mergesort and Quicksort**

#include <iostream>

using namespace std;

*//Merging algorithm used by mergeSort algorithm*

void merge(int *a*[], int *head*, int *mid*, int *tail*)

{

*//variables declaration*

    int b[10];

    int h1 = *head*, t1 = *mid*, h2 = *mid* + 1, t2 = *tail*;

    int index = h1;

*/\**

*For loop until one of the half array is*

*completely merged into array b.*

*\*/*

    for (; (h1 <= t1) && (h2 <= t2); index++)

    {

*//Comparing elements between two half array*

        if (*a*[h1] < *a*[h2])

        {

            b[index] = *a*[h1];

            h1++;

        }

        else

        {

            b[index] = *a*[h2];

            h2++;

        }

    }

*//To push the remaining elements into array b*

    for (; h1 <= t1; h1++, index++)

    {

        b[index] = *a*[h1];

    }

*//To push the remaining elements into array b*

    for (; h2 <= t2; h2++, index++)

    {

        b[index] = *a*[h2];

    }

*//Copy array b to array a*

    for (index = *head*; index <= *tail*; index++)

    {

*a*[index] = b[index];

    }

}

*//mergeSort Function which calls recursive functions.*

void mergeSort(int *a*[], int *head*, int *tail*)

{

*/\**

*If the array is more than 2 elements,*

*then run the below statements.*

*It will calls recursive function which*

*splits the array into halves.*

*Then call the merge function to sort it*

*and merge them.*

*\*/*

    if (*head* < *tail*)

    {

        int mid = (*head* + *tail*) / 2;

        mergeSort(*a*, *head*, mid);

        mergeSort(*a*, mid + 1, *tail*);

        merge(*a*, *head*, mid, *tail*);

    }

}

*//Partion function used by quickSort*

int partition(int *a*[], int *head*, int *tail*)

{

*//variables declaration*

    int last1 = *head*, i = *head* + 1, pivot = *head*;

*/\**

*While the array is larger than 1 element,*

*it will then run the statements inside*

*the while loop.*

*\*/*

    while (i <= *tail*)

    {

*/\**

*The algorithm below is doing the partition*

*of S1 and S2.*

*\*/*

        if (*a*[i] < *a*[pivot])

        {

            last1++;

            if (i != last1)

            {

                swap(*a*[i], *a*[last1]);

            }

        }

        i++;

    }

*/\**

*Swap the head element with the last 1 element*

*and point the pivot to the index of last1 element*

*\*/*

    swap(*a*[*head*], *a*[last1]);

    pivot = last1;

*//Return the index of pivot*

    return pivot;

}

*//quickSort function which calls the recursive functions*

void quickSort(int *a*[], int *head*, int *tail*)

{

*//variable declaration*

    int pivot;

*/\**

*If the array is more than 1 element, then run the*

*statements inside the if clause*

*\*/*

    if (*head* < *tail*)

    {

*/\**

*Set the pivot index from the returned value*

*by partition function*

*\*/*

        pivot = partition(*a*, *head*, *tail*);

        quickSort(*a*, *head*, pivot - 1);

        quickSort(*a*, pivot + 1, *tail*);

    }

}

*//swap function*

void swap(int &*l*, int &*r*)

{

    int temp = *r*;

*r* = *l*;

*l* = temp;

}

int main()

{

*//variables declaration*

    int arr[10] = {24, 53, 66, 11, 44, 84, 25, 102, 32, 3};

    int tmpArr[10], a = 0, b = 5, n = 0;

*//Print out the array before sorting*

    cout << "The array before sorting = ";

    for (int i = 0; i < 10; i++)

    {

        cout << arr[i] << " ";

    }

    cout << endl;

*//Calls mergeSort on the first half of the array*

    mergeSort(arr, 0, 4);

*//Print out the first half of the array after Merge Sort*

    cout << "The first part of the array after Merge Sort = ";

    for (int i = 0; i < 5; i++)

    {

        cout << arr[i] << " ";

    }

    cout << endl;

*//Calls quickSort on the last half of the array*

    quickSort(arr, 5, 9);

*//Print out the last half of the array after Quick Sort*

    cout << "The second part of the array after Quick Sort = ";

    for (int i = 0; i < 5; i++)

    {

        cout << arr[i + 5] << " ";

    }

    cout << endl;

*//Print out the array before doing the final sorting*

    cout << "The array before sorting the both part = ";

    for (int i = 0; i < 10; i++)

    {

        cout << arr[i] << " ";

    }

    cout << endl;

*//Insert both halves of the array into tmpArr by comparing the elements*

    while (n < 10)

    {

        if (a < 5 && b < 10 && arr[a] <= arr[b])

        {

            tmpArr[n] = arr[a];

            a++;

            n++;

        }

        else if (a < 5 && b < 10 && arr[a] > arr[b])

        {

            tmpArr[n] = arr[b];

            b++;

            n++;

        }

        else if (a == 5)

        {

            tmpArr[n] = arr[b];

            b++;

            n++;

        }

        else

        {

            tmpArr[n] = arr[a];

            a++;

            n++;

        }

    }

*//Copy the elements from tmpArr to arr*

    for (int i = 0; i < 10; i++)

    {

        arr[i] = tmpArr[i];

    }

*//Print out the array after sorting*

    cout << "The array after sorting = ";

    for (int i = 0; i < 10; i++)

    {

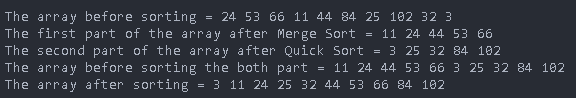
        cout << arr[i] << " ";

    }

    cout << "\n\n";

    return 1;

}

**Output**

**2. Stacks**

#include <iostream>

using namespace std;

*// Class Stack declaration*

class Stack

{

public:

*//Constructor*

    Stack() { values = new double[5]; };

*//Functions and variables declaration*

    bool isEmpty();

    double Top();

    void Push(double);

    double Pop();

    void displayStack();

private:

    int top = -1;

    double \*values;

};

*//isEmpty Function*

bool Stack::isEmpty()

{

*//Check if the stack is empty or not.*

    return top == -1;

}

*//Top Function*

double Stack::Top()

{

*/\**

*If the top index is -1, then there is no element inside the stack.*

*Else return the top element in the stack.*

*\*/*

    if (top == -1)

    {

        cout << "There is no element inside the stack.\n";

        return -1;

    }

    else

        return values[top];

}

*//Push Function*

void Stack::Push(double *x*)

{

*/\**

*If the stack is not full, then push the value x*

*into the stack.*

*Else print out that the stack is full.*

*\*/*

    if (top + 1 < 5)

    {

        values[top + 1] = *x*;

        top++;

    }

    else

    {

        cout << "The stack is full.\nCan't push anymore element into the stack.\n";

    }

}

*//Pop Function*

double Stack::Pop()

{

*/\**

*If the stack is not empty, then pop the top element*

*and return the value of it.*

*Else print out that it is empty.*

*\*/*

    if (!isEmpty())

    {

        return values[top--];

    }

    else

    {

        cout << "The stack is empty.\nCan't pop the top element from the stack.\n";

        return -1;

    }

}

*//displayStack Function*

void Stack::displayStack()

{

*/\**

*If the stack is empty, then print out Empty.*

*Else print out all the elements in the stack.*

*\*/*

    if (isEmpty())

    {

        cout << "Empty\n";

    }

    else

    {

        for (int i = 0; i <= top; i++)

        {

            cout << values[i] << " ";

        }

    }

    cout << endl;

}

int main()

{

*//variable declaration*

    Stack st, tmpst;

    double tmp;

*//Pushing all the elements into stack st*

    cout << "(1) Pushing the elements 25, 10, 15, 20, 30 into the stack st.\n\n";

    st.Push(25);

    st.Push(10);

    st.Push(15);

    st.Push(20);

    st.Push(30);

    cout << "(2) The process of sorting elements of stack st is:\n\n";

*/\**

*Algorithm which will arrange the stacks in ascending order*

*which means the bottom element is the largest element*

*and the top element is the smallest element in the stack.*

*The algorithm will stop once stack st is empty.*

*\*/*

*/\**

*While the stack st is not empty, the algorithm*

*will first pop out the top element from stack st*

*and put it into tmp variable.*

*\*/*

    while (!st.isEmpty())

    {

        tmp = st.Pop();

        cout << "\tst: Pop out " << tmp << " from the stack st.\n";

*/\**

*If the stack tmpst is empty, then it will push the value in tmp*

*into stack tmpst.*

*If the stack tmpst is not empty, it will compare the top element*

*in stack tmpst with the value in tmp.*

*If the top element in stack tmpst is larger than the value in tmp,*

*it will then push the top element in stack tmpst into stack st.*

*This will loops until the top element is smaller or equal to the*

*value in tmp or the stack tmpst is empty.*

*\*/*

        while (!tmpst.isEmpty())

        {

            cout << "\t\ttmpst: The top element of tmpst = " << tmpst.Top() << endl;

            if (tmpst.Top() > tmp)

            {

                cout << "\t\t\tSince " << tmpst.Top() << ">" << tmp << " , tmpst: Pop out " << tmpst.Top() << " from the stack tmpst.\n";

                cout << "\t\t\tst: Pushing " << tmpst.Top() << " into the stack st.\n";

                st.Push(tmpst.Pop());

            }

            else

            {

                cout << "\t\t\tSince " << tmpst.Top() << "<" << tmp << " , exiting the loop\n";

                break;

            }

        }

*//Then it will push the value in tmp into stack tmpst.*

        cout << "\t\ttmpst: Pushing " << tmp << " into stack tmpst.\n\n";

        tmpst.Push(tmp);

    }

*/\**

*The stack tmpst will push all its elements into*

*st stack until stack tmpst is empty.*

*\*/*

    while (!tmpst.isEmpty())

    {

        st.Push(tmpst.Pop());

    }

    cout << "(3) The sorting of stack st ends.\n\n";

    cout << "(4) The sequence of elements in stack st (from bottom to top): ";

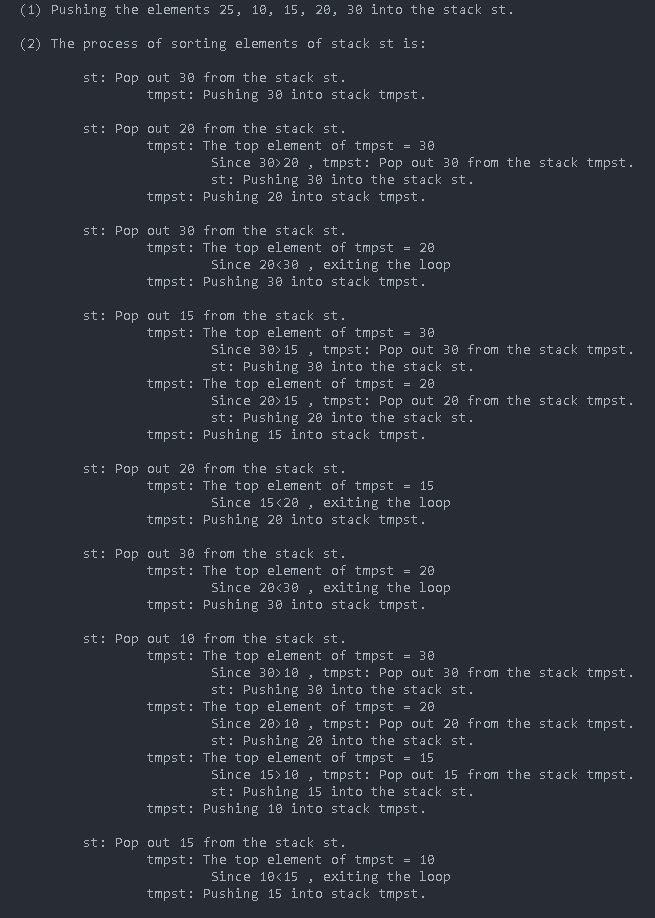
*//Displaying the stack st*

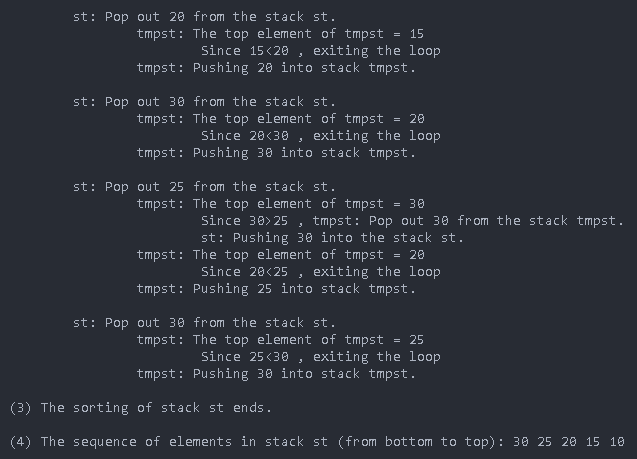
    st.displayStack();

    cout << endl;

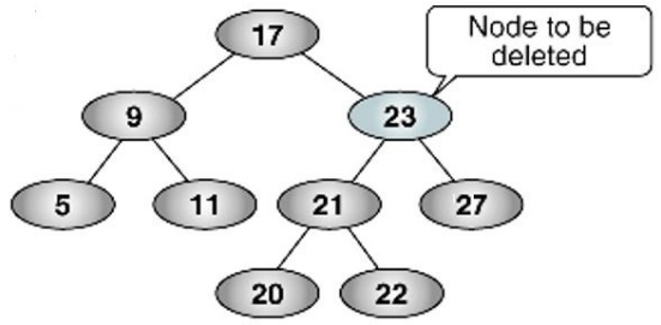
    return 1;

}

**Output**

****

**3. Trees (Binary Search Trees)**

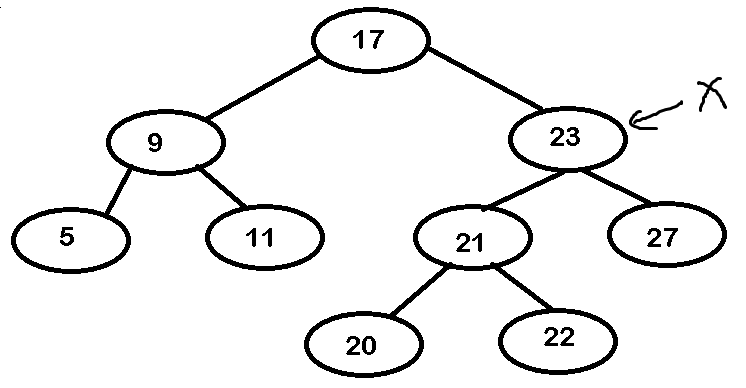
****

We are required to delete node 23.

**Step 1**

We will find the node 23 in the BST by using In-Order traversal which follows the LNR rule.

5 -> 9 -> 11 -> 17 -> 20 -> 21 -> 22 -> 23

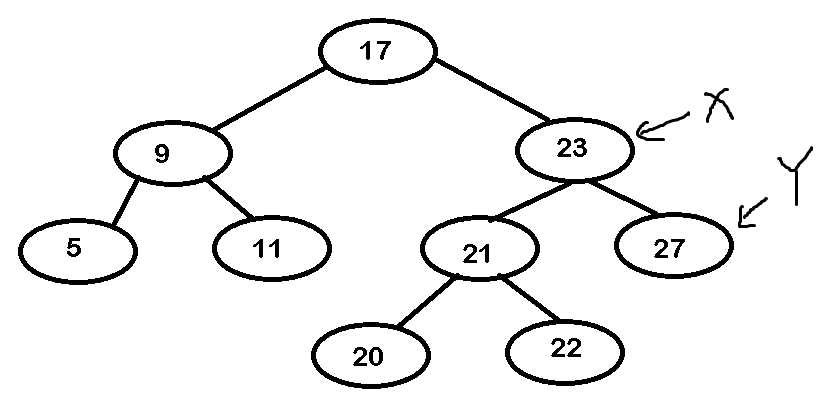
**Step 2**

Then we will put node 23 as X.

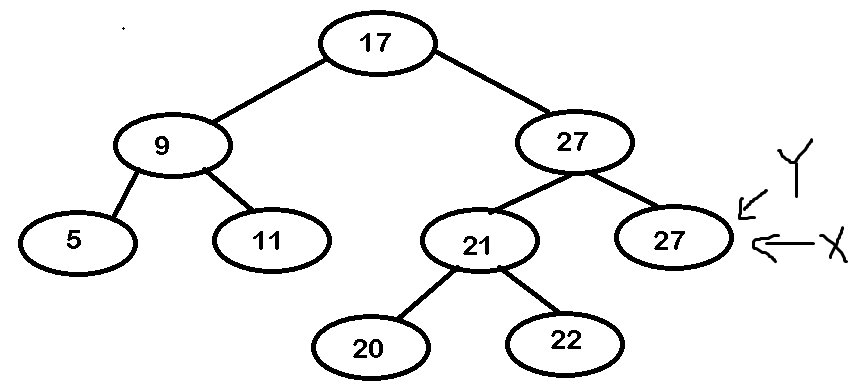
**Step 3**

We will then find the next node after node 23 by using In-Order traversal.

23 -> 27

**Step 4**

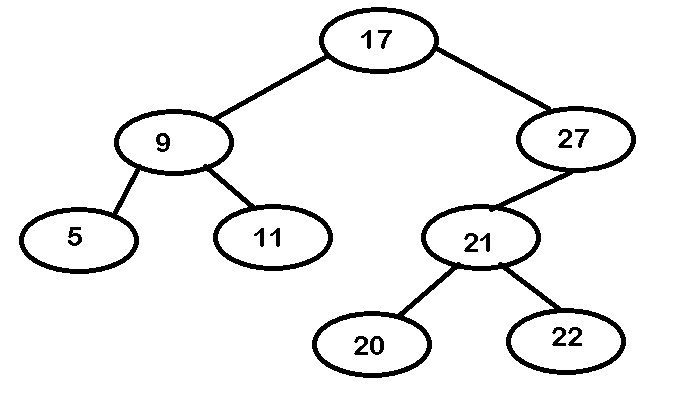
Then we will put node 27 as Y.

**Step 5**

After that, we then copy the data from Y to X and point X to Y

X->data = Y->data;

X = Y;

**Step 6**

We will then delete the node that is pointed by Y by setting the right child of parent as null and free the deleted node.

Free X;

**APPENDIX 1**

**Marking Rubrics**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Component Title** | Mergesort and Quicksort | | | | | **Percentage (%)** | **40** | |
| **Criteria** | **Score and Descriptors** | | | | | | **Weight (%)** | **Marks** |
| **(40-30)** | **(30-20)** | **(20-15)** | **(15-10)** | **(10-0)** | |
| **Mergesort and Quicksort** | **Code free of errors.**  **Detailed Comments. Output/results provided and explain algorithm with given input.**  **Using only O(n) algorithm other than then mentioned algorithms** | **Code free of errors.**  **Output/results provided and algorithm with given input is not explained enough.** | **Few mistakes in the coding. Missing some tasks or Output/results provided or algorithm with given input is not well explained.** | **Few mistakes in the coding. Missing some tasks or Output/results provided or algorithm is not explained.** | **Error in code. Missing some tasks or Output/results are not provided or algorithm is not explained.** | | **40** |  |
| **TOTAL** | | | | | | | **40** |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Component Title** | **Stacks** | | | | | **Percentage (%)** | **40** | |
| **Criteria** | **Score and Descriptors** | | | | | | **Weight (%)** | **Marks** |
| **(40-30)** | **(30-20)** | **(20-15)** | **(15-10)** | **(10-0)** | |
| **C++ program and Results, Algorithm explained with given input of set (25, 10, 15, 20, 30)** | **Code free of errors.**  **Detailed Comments. Output/results provided and explain algorithm with given input.** | **Code free of errors.**  **Output/results provided and algorithm with given input is not explained enough.** | **Few mistakes in the coding. Missing some tasks or Output/results provided or algorithm with given input is not well explained.** | **Few mistakes in the coding. Missing some tasks or Output/results provided or algorithm is not explained.** | **Error in code. Missing some tasks or Output/results are not provided or algorithm is not explained.** | | **40** |  |
| **TOTAL** | | | | | | | **40** |  |

**APPENDIX 2**

**Marking Rubrics**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Component Title** | **Trees (Binary Search Trees)** | | | | | **Percentage (%)** | **20** | |
| **Criteria** | **Score and Descriptors** | | | | | | **Weight (%)** | **Marks** |
| **(20-15)** | **(15-10)** | **(10-5)** | **(5-0)** |  | |
| **Demonstrate BST reconstruction process after deleting a node** | **Demonstrated all steps of reconstruction of new BST, with all explanations** | **Demonstrated all steps of reconstruction of new BST, with little explanations** | **Demonstrated all steps of reconstruction of new BST, with no explanations** | **Demonstrated few steps of reconstruction of new BST,**  **with no explanations** |  | | **20** |  |
| **TOTAL** | | | | | | | **20** |  |

*Note to students: Please print out and attach this appendix together with the submission of coursework*