STES'

SINHGAD COLLEGE OF ENGINEERING

VADGAON (BK), PUNE

SINHGAD TECHNICAL EDUCATION SOCIETY -PUNE

तमसो मा ज्योतिर्गमय

Sinhgad Institutes

DEPARTEMENT OF COMPUTER ENGINEERING

LAB MANUAL

Second Year of Computer Engineering (2019 Course)

210256: DATA STRUCTURES & ALGORITHMS LAB

Lab Scheme:	Credit	Examination Scheme:
PR: 04 Hours/Week	02	TW: 25 Marks PR: 25 Marks

Suggested List of Laboratory Assignments

GROUP B

5. A book consists of chapters, chapters consist of sections and sections consist of subsections. Construct a tree and print the nodes. Find the time and space requirements of your method.

6. Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree –
   i. Insert new node
   ii. Find number of nodes in longest path
   iii. Minimum data value found in the tree
   iv. Change a tree so that the roles of the left and right pointers are swapped at every node
   v. Search a value

7. For given expression eg. a-b*c-d/e+f construct inorder sequence and traverse it using postorder traversal (non recursive).

8. Read for the formulas in propositional calculus. Write a function that reads such a formula and creates its binary tree representation. What is the complexity of your function?

9. Convert given binary tree into threaded binary tree. Analyze time and space complexity of the algorithm.

10. Consider threading a binary tree using preorder threads rather than inorder threads. Design an algorithm for traversal without using stack and analyze its complexity.

11. A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Binary Search Tree for implementation.

12. Implement the Heap/Shell sort algorithm implemented in Java demonstrating heap/shell data structure with modularity of programming language.

GROUP B

13.

10. There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight takes to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Justify the storage representation used.

11. You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.

12. Tour operator organizes guided bus trips across the Maharashtra. Tourists may have different preferences. Tour operator offers a choice from many different routes. Every day the bus moves from starting city S to another city F as chosen by client. On this way, the tourists can see the sights alongside the route travelled from S to F. Client may have preference to choose route. There is a restriction on the routes that the tourists may choose from, the bus has to take a short route from S to F or a route having one distance unit longer than the minimal distance. Two routes from S to F are considered different if there is at least one road from a city A to a city B which is part of one route, but not of the other route.

13. Consider the scheduling problem. n tasks to be scheduled on single processor. Let t1, ..., tn be durations required to execute on single processor is known. The tasks can be executed in any order but one task at a time. Design a greedy algorithm for this problem and find a schedule that minimizes the total time spent by all the tasks in the system. (The time spent by one is the sum of the waiting time of task and the time spent on its execution.)

GROUP C

14. Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client‘s telephone number.

15. Implement all the functions of a dictionary (ADT) using hashing.
Data: Set of (key, value) pairs, Keys are mapped to values, Keys must be comparable, Keys must be unique
Standard Operations: Insert(key, value), Find(key), Delete(key)

16. For given set of elements create skip list. Find the element in the set that is closest to some given value.

17. The symbol table is generated by compiler. From this perspective, the symbol table is a set of

name-attribute pairs. In a symbol table for a compiler, the name is an identifier, and the attributes might include an initial value and a list of lines that use the identifier.

Perform the following operations on symbol table:

i. Determine if a particular name is in the table

ii. Retrieve the attributes of that name

iii. Modify the attributes of that name

iv. Insert a new name and its attributes

v. Delete a name and its attributes

GROUP D

18. Given sequence k = k1 <k2 < … < kn of n sorted keys, with a search probability pi for each key ki . Build the Binary search tree that has the least search cost given the access probability for each key.

19. A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Height balance tree and find the complexity for finding a keyword

GROUP E

20. To create ADT that implements the SET concept. a. Add (newElement) -Place a value into the set b. Remove (element) Remove the value c. Contains (element) Return true if element is in collection d. Size () Return number of values in collection Iterator () Return an iterator used to loop over collection e. Intersection of two sets, f. Union of two sets, g. Difference between two sets, h. Subset

21. Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject. Use heap data structure. Analyze the algorithm.

GROUP F

22. Assume we have two input and two output tapes to perform the sorting. The internal memory can hold and sort m records at a time. Write a program in java for external sorting. Find out time complexity.

23. Department maintains a student information. The file contains roll number, name, division and address. Allow user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed.

	If it is, then the system displays the student details. Use sequential file to main the data.
24.	Company maintains employee information as employee ID, name, designation and salary. Allow user to add, delete information of employee. Display information of particular employee. If employee does not exist an appropriate message is displayed. If it is, then the system displays the employee details. Use index sequential file to maintain the data.
GROUP G	
25.	Implement the Heap/Shell sort algorithm implemented in Java demonstrating heap/shell data structure with modularity of programming language.
26.	Any application defining scope of Formal parameter, Global parameter, Local parameter accessing mechanism and also relevance to private, public and protected access. Write a Java program which demonstrates the scope rules of the programming mechanism.
27.	Write a Java program which will demonstrate a concept of Interfaces and packages: In this assignment design and use of customized interfaces and packages for a specific application are expected.
28.	Write a Java program which will demonstrate a concept of cohesion and coupling of the various modules in the program.
29.	Write a program on template and exception handling in Java: in this assignment multiple templates are to be designed as a pattern and these patterns to be used to take decisions.
30.	Write a Java program for the implementation of different data structures using JAVA collection libraries (Standard toolkit library): at least 5 data structures are used to design a suitable application.
31.	Design a mini project using JAVA which will use the different data structure with or without Java collection library and show the use of specific data structure on the efficiency (performance) of the code.

Assignment List

Sr. No.	Assignment No. from Syllabus	Title
Group A		
1.	1	Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client‘s telephone number. Make use of two collision Handling Techniques (like Linear Probing, Quadratic probing etc.) and compare them using number of comparisons required to find a set of telephone number.
2.	2	Implement all the functions of a dictionary (ADT) using hashing and handle collisions using chaining with / without replacement. Data: Set of (key, value) pairs, Keys are mapped to values, Keys must be comparable, Keys must be unique. Standard Operations: Insert(key, value), Find(key), Delete(key)
GROUP B		
3.	5	A book consists of chapters, chapters consist of sections and sections consist of subsections. Construct a tree and print the nodes. Find the time and space requirements of your method
4.	6	Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree - i. Insert new node ii. Find number of nodes in longest path iii. Minimum data value found in the tree iv. Change a tree so that the roles of the left and right pointers are swapped at every node (Mirror Image). v. Search a value
5.	11 & 9	A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Binary Search Tree for implementation. OR Create In-Order Threaded Binary Search Tree. Perform Inorder traversal using threads.
Group C		
6.	13	Represent a given graph using adjacency matrix/list to perform DFS and using adjacency list to perform BFS. Use the map of the area around the college as the graph. Identify the prominent land marks as nodes and perform DFS and BFS on that.
7.	15	You have a business with several offices; you want to lease phone lines to

		connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.
Group D		
8.	18	Given sequence k = k1 <k2 < … <kn of n sorted keys, with a search probability pi for each key ki . Build the Binary search tree that has the least search cost given the access probability for each key?
9.	19	A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Height balance tree and find the complexity for finding a keyword.
Group E		
10.	22	Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject. Use heap data structure. Analyze the algorithm.
Group F		
11.	23	Department maintains a student information. The file contains roll number, name, division and address. Allow user to add, delete information of student. Display information of particular student. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to maintain the data.
12.	25	Implementation of a direct access file -Insertion and deletion of a record from a direct access file
Mini-Projects/ Case Study		
	26	Any application defining scope of Formal parameter, Global parameter, Local parameter accessing mechanism and also relevance to private, public and protected access. Write a Java program which demonstrates the scope rules of the programming mechanism.
	27	Write a Java program which will demonstrate a concept of Interfaces and packages: In this assignment design and use of customized interfaces and packages for a specific application are expected.
	29	Write a program on template and exception handling in Java: in this assignment multiple templates are to be designed as a pattern and these patterns to be used to take decisions.

GUIDELINES FOR LABORATORY CONDUCTION

Set of suggested assignment list is provided in seven groups. Each student must perform at least 13 assignments as at least

03 from group A, 02 from group B

02 from group C , 02 from group D,

01 from group E, 01 from group F

03 from group G

Operating System recommended : 64-bit Open source Linux or its derivative

Programming tools recommended: Open Source C++ Programming tool like G++/GCC

GUIDELINES FOR PRACTICAL EXAMINATION

Both internal and external examiners will jointly set problem statements. During practical assessment, the expert evaluator will give the maximum weightage to the satisfactory implementation of the problem statement. The supplementary and relevant questions may be asked at the time of evaluation to test the student's for advanced learning, understanding of the fundamentals, effective and efficient implementation.

GUIDELINES FOR ASSESSMENT

Continuous assessment of laboratory work is done based on overall performance and lab assignments performance of student. Each lab assignment assessment will assign grade/marks based on parameters with appropriate weightage. Suggested parameters for overall assessment as well as each lab assignment assessment include- *timely completion, performance, innovation, efficient codes, punctuality and neatness.*

C: Correctness, *Performance*, *Efficient Codes*	**02**	**C:** Completion Date, *Timely Completion*	**02**
R: Regularity, *Punctuality*	**02**	**P:** Presentation, *Neatness*	**06**
U: Understanding, *Innovation,*	**08**	TOTAL	**20**

GUIDELINES FOR STUDENT JOURNAL

The laboratory assignments are to be submitted by student in the form of journal. Journal consists of prologue, Certificate, table of contents, and handwritten write-up of each assignment (Title, Objectives, Problem Statement, Outcomes, software & Hardware requirements, Date of Completion, Assessment grade/marks and assessor's sign, Theory-Concept in brief, algorithm, flowchart, test cases, conclusion/analysis. Program codes with sample output of all performed assignments are to be submitted as softcopy.

<table>
<tr><th colspan="7">ASSIGNMENT NUMBER: 01</th></tr>
<tr><td colspan="3">Group A-1</td><td colspan="4"><b>Date:</b></td></tr>
<tr><td colspan="7">Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client‘s telephone number.</td></tr>
<tr><td rowspan="2">Continuous Assessment:</td><td><b>C</b></td><td><b>C</b></td><td><b>R</b></td><td><b>P</b></td><td><b>U</b></td><td><b>Total</b></td></tr>
<tr><td colspan="3"></td><td></td><td></td><td></td></tr>
<tr><td>Sign:</td><td colspan="3"><b>Date:</b></td><td colspan="3"><b>Remark:</b></td></tr>
</table>

ASSIGNMENT NUMBER: 01

Title: Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client's telephone number.

Objective:

- Understanding Hashing Concept.

Environment: Open Source C++ Programming tool like G++.

Theory:

A hash table, or a hash map, is a data structure that associates keys (names) with values (attributes).

- Look-Up Table
- Dictionary
- Cache
- Extended Array

The ideal hash table structure is merely an array of some fixed size, containing the items. A stored item needs to have a data member, called *key*, that will be used in computing the index value for the item

- Key could be an *integer*, a *string*, etc
- e.g. a name or Id that is a part of a large employee structure
- The size of the array is *TableSize*

The items that are stored in the hash table are indexed by values from *0* to *TableSize – 1.* Each key is mapped into some number in the range 0 to *TableSize – 1.* The mapping is called a *hash function.*

There are two different forms of hashing:

1. Open or external hashing
   - Allows records to be stored in unlimited space (could be hard disk)
2. Closed or internal hashing
   - Uses fixed space for storage, thus limits the size of hash table.
   -

Synonyms: If the set of keys that hash to the same location.
Collision : If the actual data that we insert into our list contains two or more synonyms , then collision occur.

Few Collision Resolution ideas :

- Separate chaining (used for open hashing)
- Some Open Addressing techniques to avoid collision.
  - Linear Probing
  - Quadratic Probing
  - Double hashing

According to problem statement, every client may have attributes like client id, client name, client address, client telephone number, etc.

We apply hash function (specifically open addressing technique like linear probing) upon client's id which is an unique attribute to quickly look up client's telephone number.
Linear Probing Technique:

- If collision (probe) occurs, alternate cells are tried until empty cell is found.
- All data is stored inside a table, large space is required.
- Whenever a collision occurs, cells are searched sequentially (with wraparound) for empty cell.
- It is easy to implement, but suffers from "*Primary Clustering*" i.e. when many keys are mapped to the same location (collision), then linear probing will not distribute keys evenly.
- These keys will be stored in the neighborhood of the location where they are mapped.
- This will lead to clustering of keys around the point of collision, and which leads to *Primary Clustering*.
- So, excessive collision is a major problem. Also, more number of searches is required to locate synonyms due to uneven distribution of keys.
-

So, in the problem statement, we consider,

Hash(key) = key % table size, where key if client id. (consider any table size)

The result is the address to store the whole client record (like client id, client name, client address, client telephone number, etc).

Algorithm:

Construction of Hash Table for Client telephone book:

Algorithm: Insert_Linear Probe (int client_id) : function returns location of a key

```
begin
        int i, j;
        j= client_id % TableSize;     // mapped location
        for(i=0; i< TableSize; i++)
        begin_for
                if(hashtable[j] is empty) then  //empty cell found
                        begin_if
                             hashtable[j] = client_record;
                             return(j);
                        end_if
                j = (j+1) % TableSize;   //next location in circular way.
        End_for
//otherwise hash table is full
return(-1);
end
```

Algorithm: Search_Linear Probe (int client_id) : function returns location of a key

```
begin
        int i, j;
        j= client_id % TableSize;    // mapped location
        for(i=0; i< TableSize; i++)
        begin_for
                if(hashtable[j].client_id ==client_id) then  //empty cell found
                        begin_if
                          // record found
                             return(j);
                        end_if
                j = (j+1) % TableSize;   //next location in circular way.
        End_for
//if record doesn't exist
return(-1);
end
```

Input:

Output:

Test Cases:

Input/input characteristics	Expected output	Observed output	Passed/failed

Conclusion:

Hash table with Linear Probing is implemented for client records.

Frequently Asked Questions (FAQ):

1) Explain Following with example:

a) Linear Probing with chaining without replacement

b) Linear Probing with chaining with replacement

2) Differentiate between Linear Probing and Quadratic Probing.

3) Discuss about Double Hashing with example.

ASSIGNMENT NUMBER: 03						
Group B-05			**Date:**			
A book consists of chapters, chapters consist of sections and sections consist of subsections. Construct a tree and print the nodes. Find the time and space requirements of your method						
Continuous Assessment:	**C**	**C**	**R**	**P**	**U**	**Total**
Sign:	**Date:**			**Remark:**		

ASSIGNMENT NUMBER: 03

Title: A book consists of chapters, chapters consist of sections and sections consist of subsections. Construct a tree and print the nodes. Find the time and space requirements of your method

Objective:

- Understand the concept of General Tree
- Concept of General tree to Binary tree Conversion
- Implementation of Binary Tree

Environment: Open Source C++ Programming tool like G++

Theory:

A tree is a nonlinear data structure, compared to arrays, linked lists, stacks and queues which are linear data structures. A tree can be empty with no nodes or a tree is a structure consisting of one node called the root and zero or one or more subtrees.

A tree has following general properties:

- One node is distinguished as a root;
- Every node (exclude a root) is connected by a directed edge from exactly one other node; A direction is: parent -> children

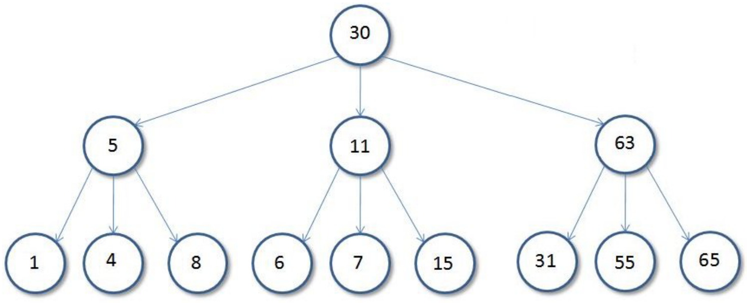


Figure 1 General Tree

30 is a root node

30 is a parent of 5, 11, 63

5, 11, 63 are children of 30

(5, 11, 63), (1, 4, 8), (6, 7, 15), (31, 55, 65) are siblings

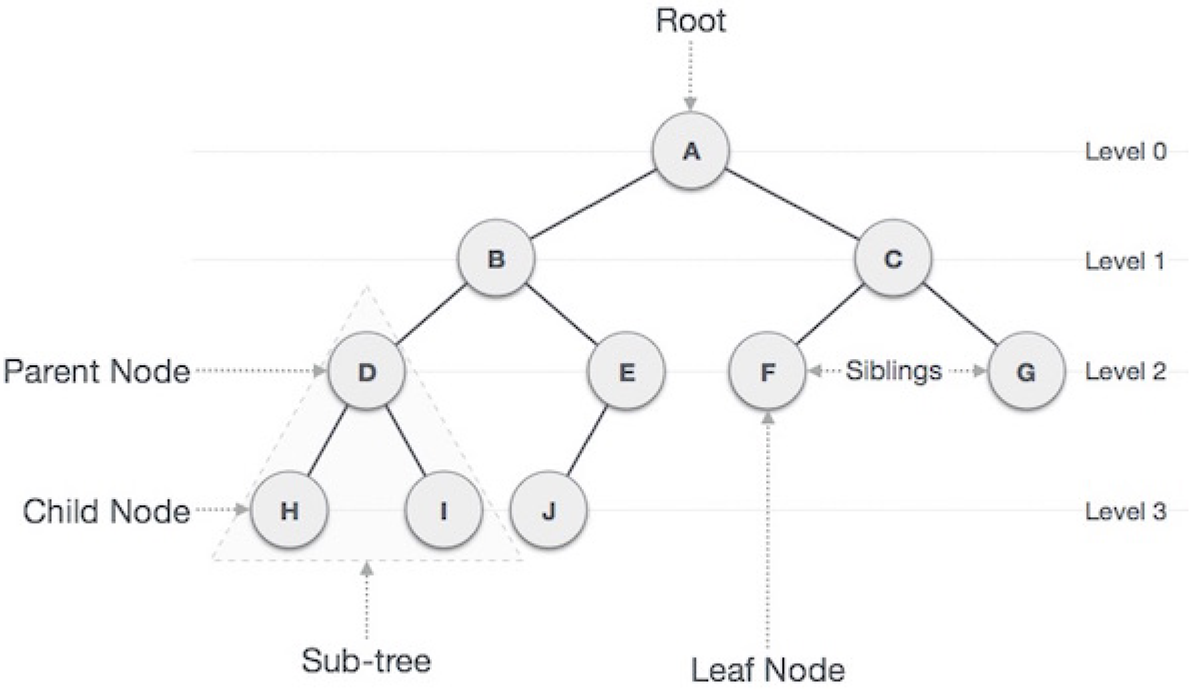
In a tree *number of edges coming in* to particular node is called In-degree of that particular node. Root node having In-degree always equals to 0 because it is the first node of tree. In a tree *number of edges going out from* particular node is called Out-degree of that particular node. Leaf node having out degree always equals to 0 because it is the last node of tree having no further sub tree.

In a tree the sum of edges coming into particular node and edges going out from particular node is called degree of that particular node. It means *Degree is the sum of in degree and out degree*. It is also called total degree of node.

Binary Tree is a special data structure used for data storage purposes. A binary tree has a special condition that each node can have a maximum of two children.

A binary tree has the benefits of both an ordered array and a linked list as search is as quick as in a sorted array and insertion or deletion operation are as fast as in linked list.

In terms of degree, a tree in which out degree of each node is less than or equal to 2 but not less than 0 is called binary tree. We can also say that a tree in which each node having either 0, 1 or 2 child is called binary tree.



Conversion of General Tree in to Binary Tree:

The process of converting general tree in to binary tree is given below:

1. Root node of general tree becomes root node of Binary Tree.
2. Now consider T_1, T_2, T_3 ... T_n are child nodes of the root node in general tree. The left most

children (T_1) of the root node in general tree become left most child of root node in the binary tree. Now Node T_2 becomes right child of Node T_1, Node T_3 becomes right child of Node T_2 and so on in binary tree

3. The same procedure of step 2 is repeated for each leftmost node in the general tree.

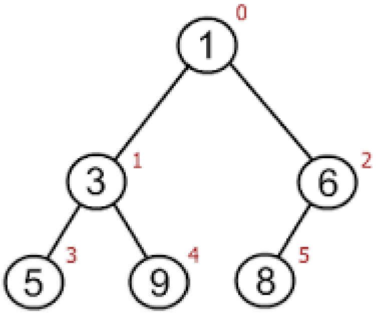
Representation of Binary Tree:

A binary tree data structure is represented using two methods:

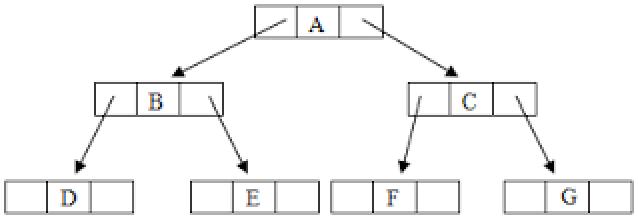
- Array Representation
- Linked List Representation

In *array representation of binary tree*, we use a *one dimensional array (1-D Array)* to represent a binary tree. To represent a binary tree of depth/height 'h' using array representation, we need one dimensional array with a maximum size of $2^{h+1} - 1$.

In *linked representation*, we use *doubly linked list* to represent a binary tree. In a double linked list, every node consists of three fields. First field for storing left child address, second for storing actual data and third for storing right child address.



1	3	6	5	9	8
0	1	2	3	4	5



Algorithm:

Construction of Binary Tree:

Input: General Tree represented in Binary Tree form for Book

Output: Traversal of Binary Tree

Test Cases:

Input/input characteristics	Expected output	Observed output	Passed/failed

Conclusion: Binary tree was constructed using linked representation and was traversed in inorder, postorder and preorder.

Frequently Asked Questions (FAQ):

1. Define the terms : General Tree, Binary Tree, internal node, leaf node, siblings, in-degree, out -degree, degree, level, depth/height, path
2. Can we convert a general tree into binary tree? What are the steps to convert a general tree into binary tree? Explain with an example.
3. Explain the binary tree representations.
4. What are the advantages and disadvantages of array representation over linked representation of binary tree?

ASSIGNMENT NUMBER: 04

Group B-06	**Date:**

Title: Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree –

i. Insert new node

ii. Find number of nodes in longest path

iii. Minimum data value found in the tree

iv. Change a tree so that the roles of the left and right pointers are swapped at every node

v. Search a value

Continuous Assessment:	**C**	**C**	**R**	**P**	**U**	**Total**

Sign:	**Date:**	**Remark:**

ASSIGNMENT NUMBER: 04

Title: Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree –

i. Insert new node

ii. Find number of nodes in longest path

iii. Minimum data value found in the tree

iv. Change a tree so that the roles of the left and right pointers are swapped at every node

v. Search a value

Objective:

- Implement various operations of a binary search tree, such as insertion, finding the smallest and largest element, mirror, etc.
- Derive the time complexities for the above operations on a binary search tree.
- Give that advantages and disadvantages of a binary search tree over other data structures
- Identify applications where a binary search tree will be useful.

Environment: Open Source C++ Programming tool like G++

Theory:

A binary tree is called a binary search tree (BST) if for every node in the tree the data elements in the node is greater than the data elements in all the nodes in the left subtree and is less than or equal to the data elements in all the nodes in the right subtree.

Consider a binary tree T. T is a binary search tree, that is, the value of the data elements in every node N in T greater than to the data elements in every node in left subtree and is less than or equal to the data elements in every node in right subtree.

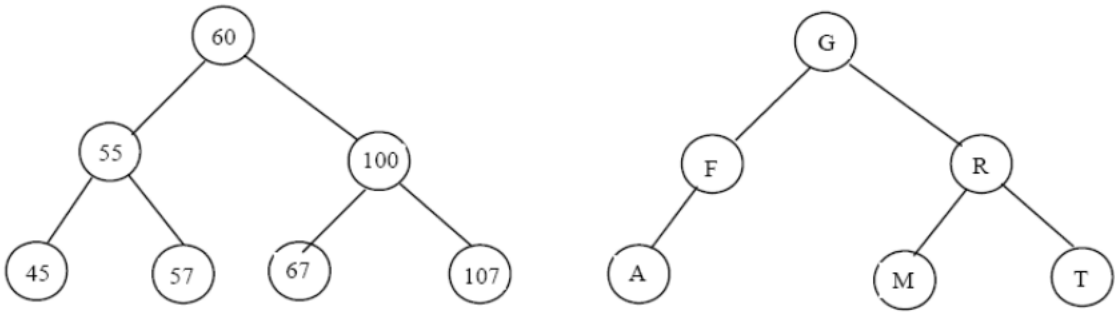


Figure 2. Binary Search Tree (BST)

Linked Representation of Binary Search Tree

A binary search tree can be represented using a set of linked nodes. Each node contains a value and two links named left and right that reference the left child and right child, respectively

Pointer to Left Subtree	Data	Pointer to Right Subtree

Figure 3. Node

The variable *root* refers to the root node of the tree. If the tree is *empty*, root is NULL.

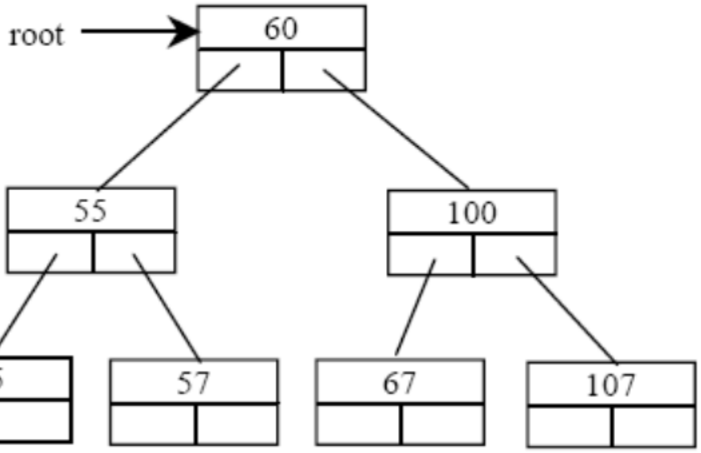


Figure 4. Linked Representation of Binary Search Tree

Algorithm:

Input: Sequence of input data in different orders like increasing order, decreasing order, random order.

Output: Sequence of output data

Test Cases:

Input/input characteristics	Expected output	Observed output	Passed/failed

Conclusion: The nature of binary tree constructed with change in the order of data was observed. The effect of this on various algorithms like insert, search and finding the number of nods in the longest path, finding minimum was noticed.

Frequently Asked Questions (FAQ):

1. Construct a binary search tree (BST) for following input data 56, 77, 22, 28, 89, 11, 70
2. State time complexities of following algorithms in BST

   2.1. Insert

   2.2. Find smallest

   2.3. Search
3. What is the limitation of BST with respect to performance?

ASSIGNMENT NUMBER: 05						
Group B-09		**Date:**				
Convert given binary tree into threaded binary tree. Analyze time and space complexity of the algorithm. Consider threading a binary tree using preorder threads rather than inorder threads. Design an algorithm for traversal without using stack and analyze its complexity						
Continuous Assessment:	C	C	R	P	U	Total
Sign:	**Date:**			**Remark:**		

ASSIGNMENT NUMBER: 05

Title: Convert given binary tree into threaded binary tree. Analyze time and space complexity of the algorithm. Consider threading a binary tree using preorder threads rather than inorder threads. Design an algorithm for traversal without using stack and analyze its complexity

Objective:

- Understanding the effective use of NULL pointers to improve the performance in tree traversal operations

Environment: Open Source C++ Programming tool like G++

Theory:

Inorder traversal of a binary tree is either done using recursion or with the use of an auxiliary stack. The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists). Such a pointer is called as a thread and must be differentiable from a tree pointer that is used to link a node to its left or right subtree.

There are two types of threaded binary trees.

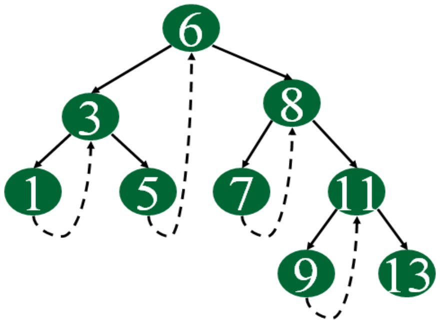
Single Threaded: Where a NULL right pointers is made to point to the inorder successor (if successor exists). Here the rightmost node in in each tree still has NULL right pointer, since it has no inorder successor. Such trees are also called as *right-in threaded* binary trees. A *left in-threaded* binary tree may be defined similarly, in which each NULL left pointer is altered to contain a thread to that node's inorder predecessor.

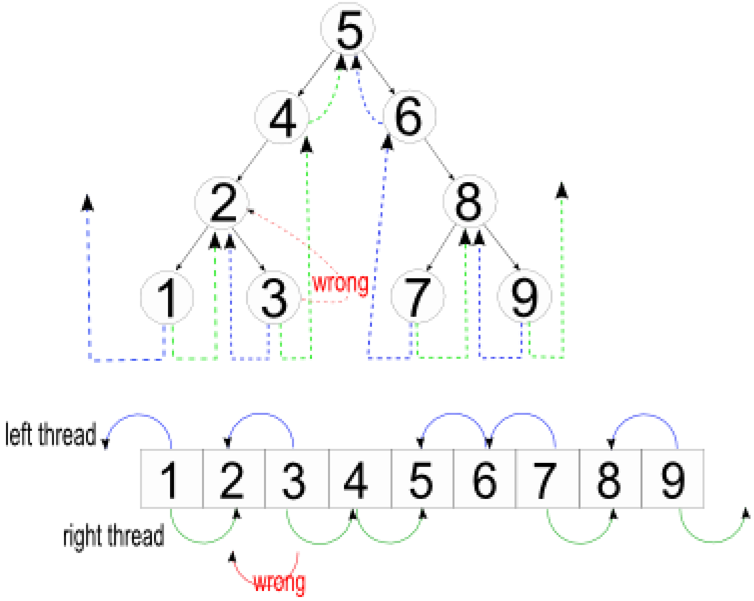
Double Threaded: Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively. Such trees are called as *in-threaded* binary trees. The predecessor threads are useful for reverse inorder traversal and postorder traversal. Thus in-thread binary tree is both left in-threaded and right in-threaded.

Right and left *pre-threaded* binary trees are those in which NULL right and left pointers of nodes are replaced by their preorder successor and predecessors respectively.

The threads are also useful for fast accessing ancestors of a node.

Following diagrams shows examples of right in-thread (Single) binary tree and in-threaded binary trees. The dotted lines represent threads.





Algorithm:

Convert Binary Tree into Treaded Binary Tree:

Solution:

- We can do the inorder traversal and store it in some queue.
- Do another inorder traversal and where ever you find a node whose right pointer is NULL, take the front element from the queue and make it the right of the current node and mark it as a thread.

Better Solution:

The problem with the earlier solution is it's using the extra space for storing the inorder traversal and using two traversals. This solution will convert binary tree into threaded binary tree in one single traversal with no extra space required.

- Do the reverse inorder traversal, means visit right child first
- In recursive call, pass additional parameter, the node visited previously
- Whenever you will find a node whose right pointer is NULL and previous visited node is not

NULL then make the right of node points to previous visited node and mark the r-thread as true.

- Important point is whenever making a recursive call to right subtree, do not change the previous visited node and when making a recursive call to left subtree then pass the actual previous visited node.

```
    public void convert(Node  *root){
        inorder(root, null);
      }

      public void inorder(Node *root, Node *previous)
    {
        if(root==null)
        {
           return;
        }
      else{
           inorder (root.right, previous);
           if(root.right==null &&  previous!=null)
           {
              root.right = previous;
              root.rightThread=true;
           }
           inorder(root.left, root);
        }
    }
```

Inorder traversal of Threaded Binary Tree:

```
  Node*  leftMostNode (Node *node){
      if(node==null){
         return null;
      }else{
         while(node->left!=null){
            node = node->left;
         }
         return node;
      }
}
void print(Node *root){
      //first go to most left node
      Node *current = leftMostNode(root);
      //now travel using right pointers
      while(current!=null){
         cout<<current->data;
         //check if node has a right thread
         if(current->rightThread)
```

```
            current = current->right;
         else // else go to left most node in the right subtree
            current = leftMostNode(current->right);
      }
}
```

Input: Sequence of numbers to construct binary tree

Output: Traversal of binary tree

Test Cases:

Input/input characteristics	Expected output	Observed output	Passed/failed

Conclusion:

The need of implicit or explicit stack in the traversal operation was found eliminated.

Frequently Asked Questions (FAQ):

1. Why do we need Threaded Binary Tree?
2. What are the Types of threaded binary trees?
3. What are the advantages of Threaded Binary Tree?

ASSIGNMENT NUMBER: 06						
Group C-13		Date:				
Write a function to get the number of vertices in an undirected graph and its edges. You may assume that no edge is input twice. i. Use adjacency list representation of the graph and find runtime of the function. ii. Use adjacency matrix representation of the graph and find runtime of the function.						
Continuous Assessment:	C	C	R	P	U	Total
Sign:	Date:			Remark:		

ASSIGNMENT NUMBER: 06

Title: Write a function to get the number of vertices in an undirected graph and its edges. You may assume that no edge is input twice.
i. Use adjacency list representation of the graph and find runtime of the function.
ii. Use adjacency matrix representation of the graph and find runtime of the function.

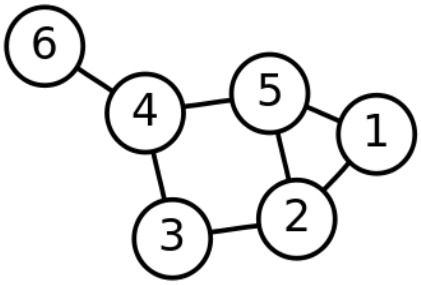
Objective:

- Understanding the representation of Adjacency Matrix and Adjacency List representation for graph creation.

Environment: Open Source C++ Programming tool like G++

Theory:

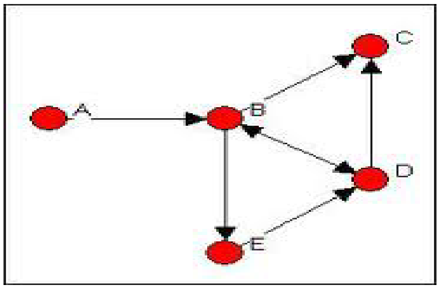
A graph G is an ordered pair (V, E) where V is the set of vertices and E is the set of edges. Each edge is associated with an unordered pair (vi , vj). The vertices Vi & vj are called the end vertices or the terminal vertices of the edge Eij.



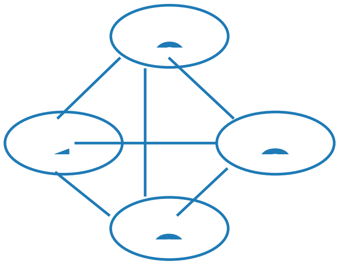
Graph theory is the study of points and lines. In particular, it involves the ways in which sets of points, called vertices, can be connected by lines or arcs, called edges. Graphs in this context differ from the more familiar coordinate plots that portray mathematical relations and functions.

Mainly two types of graph:

1) Directed Graph



2) Undirected Graph



- Storage representation:

- Adjacency matrix
- Adjacency List
- Adjacency Multilist

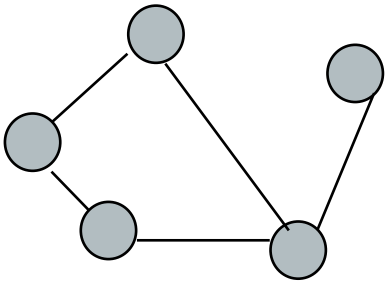
1) Adjacency matrix:- Adjacency matrix is a Squared two dimensional array. Matrix size is n x n, where n = No of vertices.An entry in a row i and column j is 1 if there is an edge incident between vertex i and vertex j , and is 0 otherwise.

   A[i][j] = 1 iff (i,j) is an edge

   A[i][j] = o iff edge (i,j) does not exist.

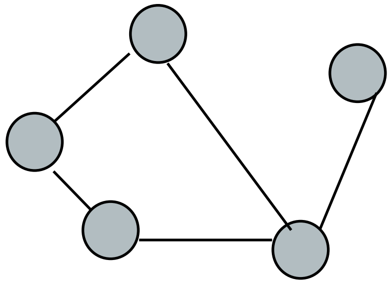
   If graph is weighted graph , then the entry 1 is replaced with the weight.

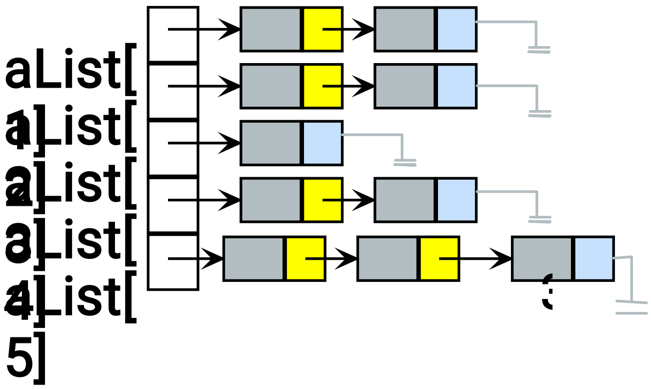
   If graph is directed , we interpret 1 stored at A[i][j] , indicating that the edge from i to j exists and not indicating whether or not the edge from j to I exists in the graph



	1	2	3	4	5
1					
2					
3					
4					
5					

2) Adjacency List:- Each adjacency list is a linked chain. Array Length = n (no. of vertices), Count of chain nodes = 2e (undirected graph), Count of chain nodes = e (digraph).





Algorithm:

Construction of Adjacency Matrix:

```
Algorithm: InsertMatrix (Source Vertex, End Vertex)
1. If type of a graph is Directed then
    1.1.    Graph[SourceVertex][DestinationVertex]=1
    1.2.    Return
2. Else
    2.1.    Graph [SourceVertex] [DestinationVertex] =1
    2.2.    Graph [DestinationVertex] [ SourceVertex] =1
End.
```

Construction of Adjacency List:

```
Algorithm: InsertNodeInList(Source Vertex, End Vertex)
    1. Create NewNode dynamically.
        1.1.  If HeadArray[Vertex] is empty then
                1.1.1. Assign NewNode to HeadArray[Vertex].
        1.2. Else
                1.2.1. Traverse till Adjacency List's end.
                1.2.2. Assign NewNode at the end of HeadArray[Vertex] list.
End.
```

Input: Number of vertices and edges. Source and destination vertices for edges.

Output: Adjacency matrix and list

Test Cases:

Input/input characteristics	Expected output	Observed output	Passed/failed

Conclusion:

Graph is represented by Adjacency Matrix and List.

Frequently Asked Questions (FAQ):

1. Explain Following with example:

1.1 Subgraph

1.2 Adjacency matrix and List

2. What are different ways to represent a graph? Give suitable example. Also differentiate within them.

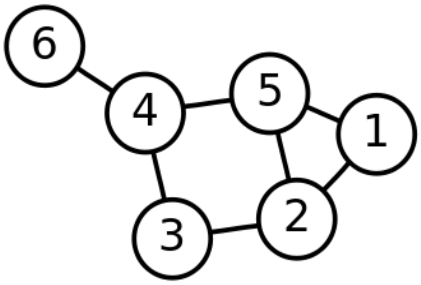
3. Explain any three real world applications of Graph?

ASSIGNMENT NUMBER: 07						
Group C-15			**Date:**			
You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.						
Continuous Assessment:	**C**	**C**	**R**	**P**	**U**	**Total**
Sign:	**Date:**			**Remark:**		

ASSIGNMENT NUMBER: 07
Title: You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.
Objective: • Understanding Minimum Spanning Tree.
Environment: Open Source C++ Programming tool like G++

Theory:

A graph G is an ordered pair (V, E) where V is the set of vertices and E is the set of edges. Each edge is associated with an unordered pair (vi , vj). The vertices Vi & vj are called the end vertices or the terminal vertices of the edge Eij.



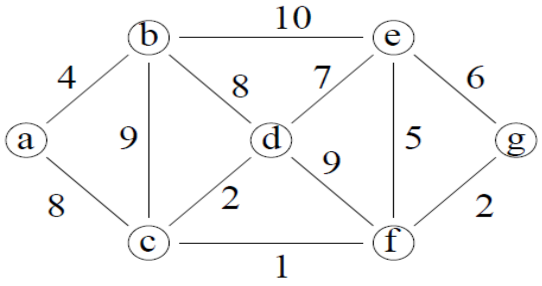
Several Offices in the statement represent vertices of a graph and the lease lines connecting them represent the edges. The objective is to connect all offices by the lease lines with minimum cost. This is the problem of creating Minimum Spanning Tree.

There are two ways to create MST:

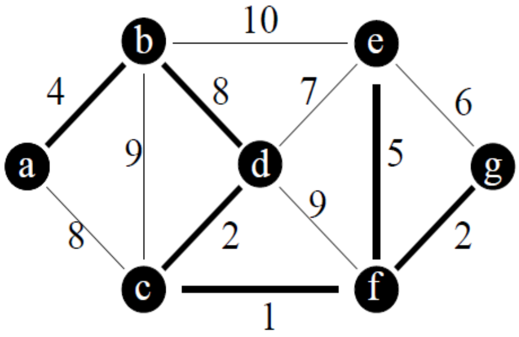
1) Prim's Algorithm

2) Kruskal's Algorithm

Prim's Algorithm:



Minimum Spanning Tree for above graph:



Algorithm:

Construction of Minimum Spanning Tree by Prim's Method:

Algorithm: InsertMatrix (Source Vertex, End Vertex)

Let the graph G=(V, E) has n vertices.

1. Choose a vertex V_1 of G. let V_T ={V_1} and E_T ={ }.
2. Choose a nearest neighbor V_i of V_T that is adjacent to V_j, V_j belongs to V_T and for which the edge (V_i, V_j) does not form a cycle with members of E_T. Add V_i to V_T and add (V_i, V_j) to E_T.
3. Repeat step 2 until | E_T | = n-1. Then V_T contains all n vertices of G and E_T contains the edges of the minimum cost spanning tree of G.

Input: Number of vertices and edges representation with Adjacency Matrix.

Output: Minimum Spanning Tree.

Test Cases:

Input/input characteristics	Expected output	Observed output	Passed/failed

Conclusion:

Minimum Spanning Tree

Frequently Asked Questions (FAQ):

1) Explain Following with example:

a) Strongly connected components

b) Spanning Tree

2) What is Minimum Spanning Tree? How is it different than the Shortest path Sequence of the Graph? Justify your answer with example.

3) Discuss about time complexities of prim's and Kruskal's algorithms to find Minimum Spanning Tree.

ASSIGNMENT NUMBER: 09

Group D - 19	**Date:**

vi. Title:

A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Height balance tree and find the complexity for finding a keyword

Continuous Assessment:	C	C	R	P	U	**Total**

Sign:	**Date:**	**Remark:**

ASSIGNMENT NUMBER: 09

Title: A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Height balance tree and find the complexity for finding a keyword.

Objective:

To understand importance of searching time in search.

To learn concept Height balance tree.
Find out time complexity for finding a keyword

Environment: Open Source C++ Programming tool like G++

Theory:

The disadvantage of a binary search tree is that its height can be as large as N-1. This means that the time needed to perform insertion and deletion and many other operations can be O(N) in the worst case. A binary tree with N node has height at least Θ(log N). Thus, our goal is to keep the height of a binary search tree O(log N). Such trees are called balanced binary search trees. Examples are AVL tree, red-black tree.

AVL tree is a self balanced binary search tree. That means, an AVL tree is also a binary search tree but it is a balanced tree. A binary tree is said to be balanced, if the difference between the hieghts of left and right subtrees of every node in the tree is either -1, 0 or +1. In other words, a binary tree is said to be balanced if for every node, height of its children differ by at most one. In an AVL tree, every node maintains a extra information known as balance factor. The AVL tree was introduced in the year of 1962 by G.M. Adelson-Velsky and E.M. Landis.

Balance factor of a node is the difference between the heights of left and right subtrees of that node. The balance factor of a node is calculated either height of left subtree - height of right subtree.

In the following explanation, we are calculating as follows.

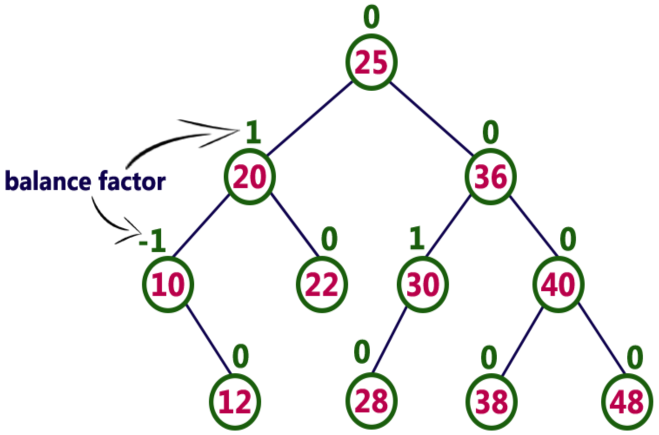


Figure 1. AVL Tree

Rotation operations are used to make a tree balanced. Rotation is the process of moving the nodes to either left or right to make tree balanced. There are four rotations and they are classified into two types.

AVL Rotations

To balance itself, an AVL tree may perform the following four kinds of rotations –

Left rotation

Right rotation

Left-Right rotation

Right-Left rotation

The first two rotations are single rotations and the next two rotations are double rotations. To have an unbalanced tree, we at least need a tree of height 2. With this simple tree, let's understand them one by one.

Left Rotation

If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation –

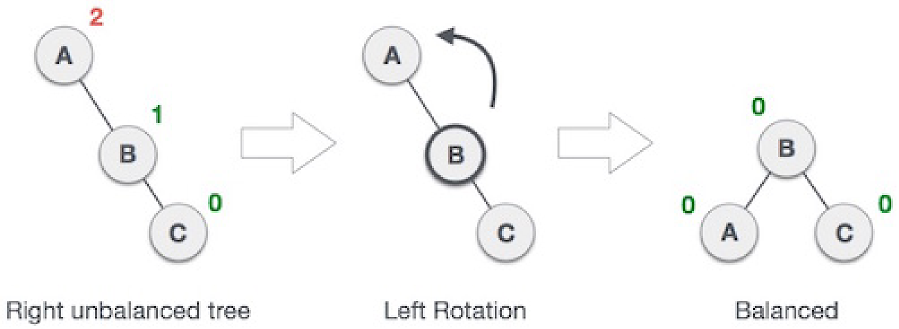


Figure 2: Left rotation

In our example, node A has become unbalanced as a node is inserted in the right sub tree of A's right sub tree. We perform the left rotation by making A the left-sub tree of B.

Right Rotation

AVL tree may become unbalanced, if a node is inserted in the left sub tree of the left sub tree. The tree then needs a right rotation.

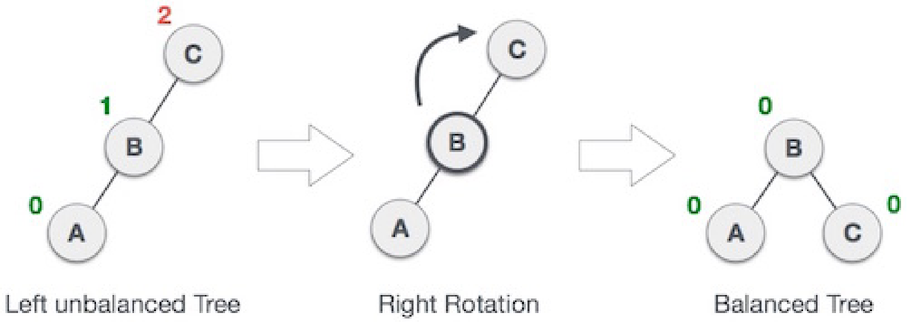


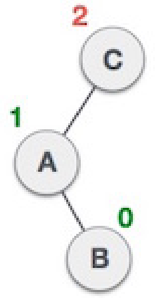
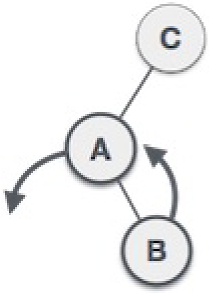
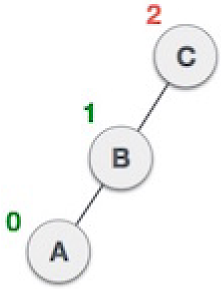
Figure 3: Right rotation

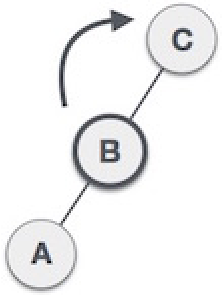
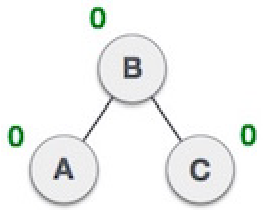
As depicted, the unbalanced node becomes the right child of its left child by performing a right rotation.

Left-Right Rotation

Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation. A left-right rotation is a combination of left rotation followed by right rotation.

Example:

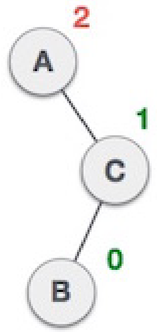
State	Action
	A node has been inserted into the right sub tree of the left sub tree. This makes **C** an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.
	We first perform the left rotation on the left sub tree of **C**. This makes **A**, the left sub tree of **B**.
	Node **C** is still unbalanced, however now, it is because of the left-sub tree of the left-sub tree.

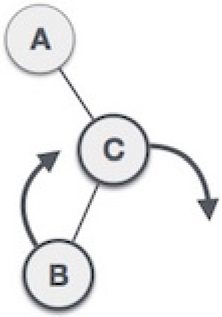
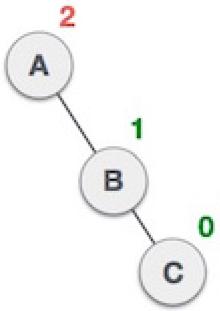
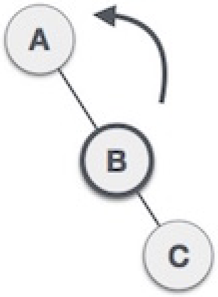
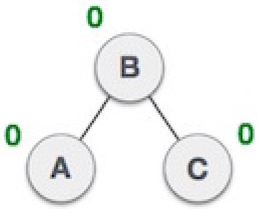
	We shall now right-rotate the tree, making **B** the new root node of this sub tree. **C** now becomes the right sub tree of its own left sub tree.
	The tree is now balanced.

Right-Left Rotation

The other type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

Example-

State	Action
	A node has been inserted into the left sub tree of the right sub tree. This makes **A**, an unbalanced node with balance factor 2.

	First, we perform the right rotation along **C** node, making **C** the right sub tree of its own left sub tree **B**. Now, **B** becomes the right sub tree of **A**.
	Node **A** is still unbalanced because of the right sub tree of its right sub tree and requires a left rotation.
	A left rotation is performed by making **B** the new root node of the sub tree. **A** becomes the left sub tree of its right sub tree **B**.
	The tree is now balanced.

Algorithm:

1. Rotate Right

```
AVLNode *AVL::rotate_right(AVLNode *x)
{
cout<<"\nright Rotate";
AVLNode *y;
y=x->left;
x->left=y->right;
y->right=x;
x->ht=height(x);
y->ht=height(y);
return(y);
}
```

2. Rotate Left

```
AVLNode *AVL::rotate_left(AVLNode *x)
 {
  cout<<"\nleft Rotate";
  AVLNode *y;
  y=x->right;
  x->right=y->left;
  y->left=x;
  x->ht=height(x);
  y->ht=height(y);
  return(y);
 }
```

Input: Keyword to searched its meaning

Output: Meaning of keyword

Test Cases:

Input/input characteristics	Expected output	Observed output	Passed/failed

Conclusion:

Thus we have implemented insertion and deletion on Dictionary which uses Height balance Binary Search Tree (BST) Data structure with less time complexity O(h).

FAQ:

What is AVL? What is search time complexity for AVL

1. What is the purpose of AVL trees?
2. What is left rotation?
3. Create AVL tree of given sequence-

   MAR,MAY,NOV,AUG,APR,JAN,DEC,JUL,FEB,JUN,OCT,SEP

ASSIGNMENT NUMBER: 10

Group E-22	**Date:**

Title: Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject. Use heap data structure. Analyze the algorithm.

Continuous Assessment:	C	C	R	P	U	**Total**
Sign:	**Date:**			**Remark:**		

ASSIGNMENT NUMBER: 10

Title: Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject. Use heap data structure. Analyze the algorithm.

Objective:

- Implement various operations of a heap data structure, such as insertion, finding the smallest and largest marks obtained by the students etc.
- Derive the time complexities for the above operations on a heap data structure.
- Give that advantages and disadvantages of a heap data structure.
- Identify applications where a heap data structure will be useful.

Environment: Open Source C++ Programming tool like G++

Theory:

A large area of memory from which the programmer can allocate blocks as needed, and deal locate them (or allow them to be garbage collected) when no longer needed. (or) A balanced, left-justified binary tree in which no node has a value greater than the value in its parent.

A balanced binary tree is left-justified if:

- all the leaves are at the same depth, or
- all the leaves at depth n+1 are to the left of all the nodes at depth n.

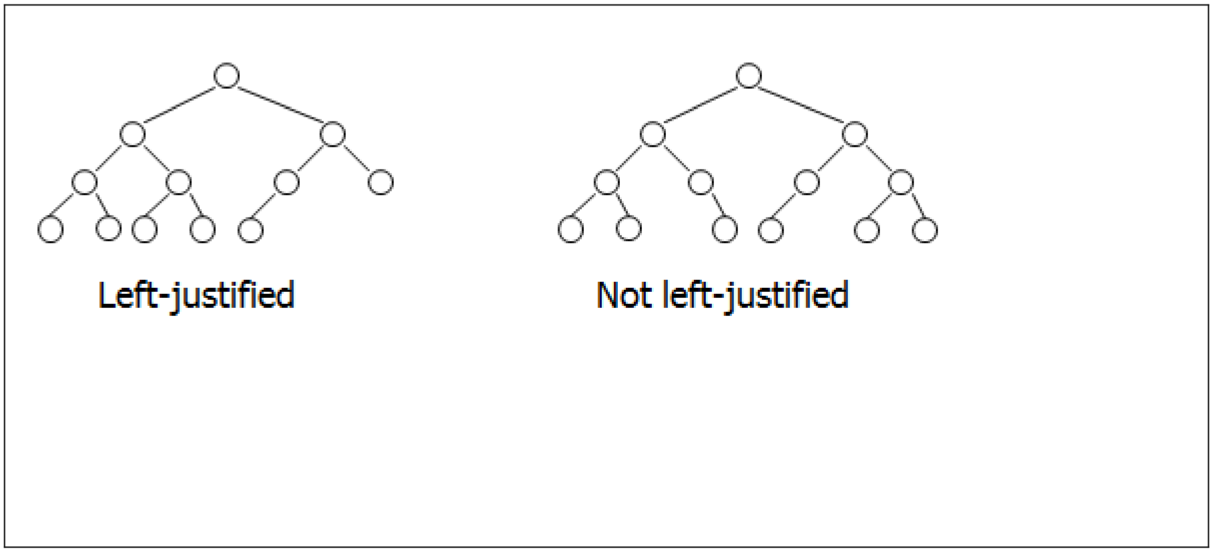


Figure 1. Left-justified and Not left-justified trees

Heap Property:

All nodes are either [greater than or equal to] or [less than or equal to] each of its children. If the parent nodes are greater than their children, heap is called a Max-Heap, and if the parent nodes are smaller than their child nodes, heap is called Min-Heap.

Array Representation of heap data structure:

A complete binary tree can be uniquely represented by storing its level order traversal in an array.

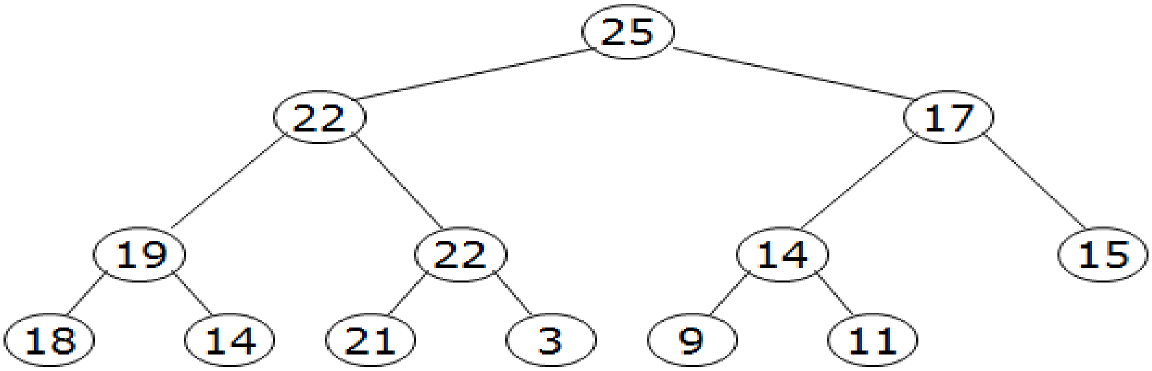
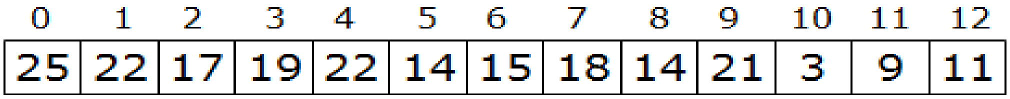


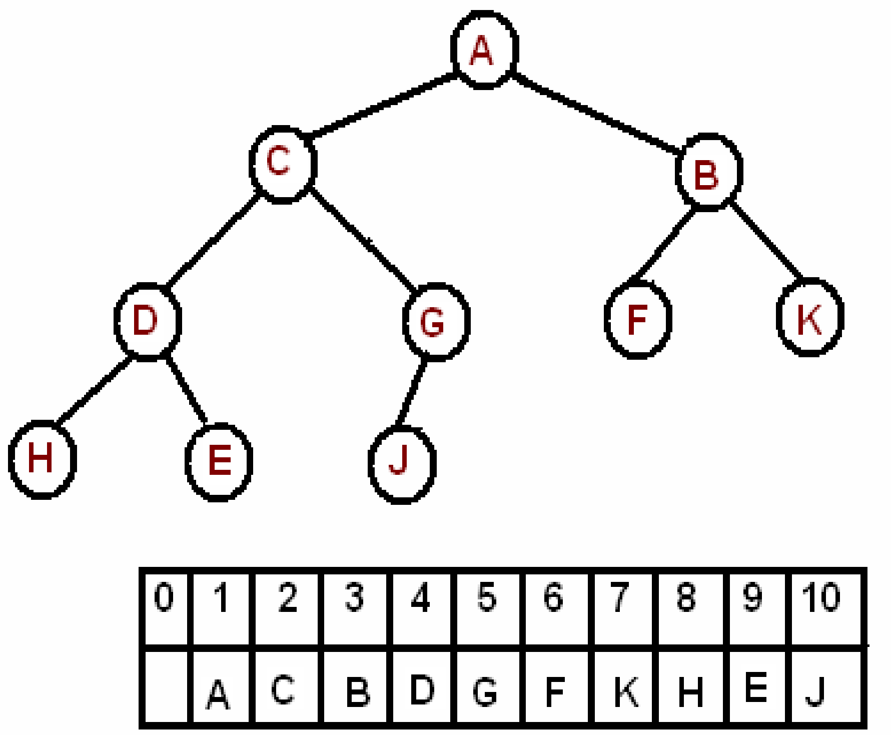
Figure 2. Array Representation of heap data structure



0	1	2	3	4	5	6	7	8	9	10	11	12
25	22	17	19	22	14	15	18	14	21	3	9	11

- Notice:
  - The left child of index i is at index 2*i+1
  - The right child of index i is at index 2*i+2
  - Example: the children of node 3 (19) are 7 (18) and 8 (14)

(OR)



0	1	2	3	4	5	6	7	8	9	10
	A	C	B	D	G	F	K	H	E	J

Figure 2. Array Representation of heap data structure

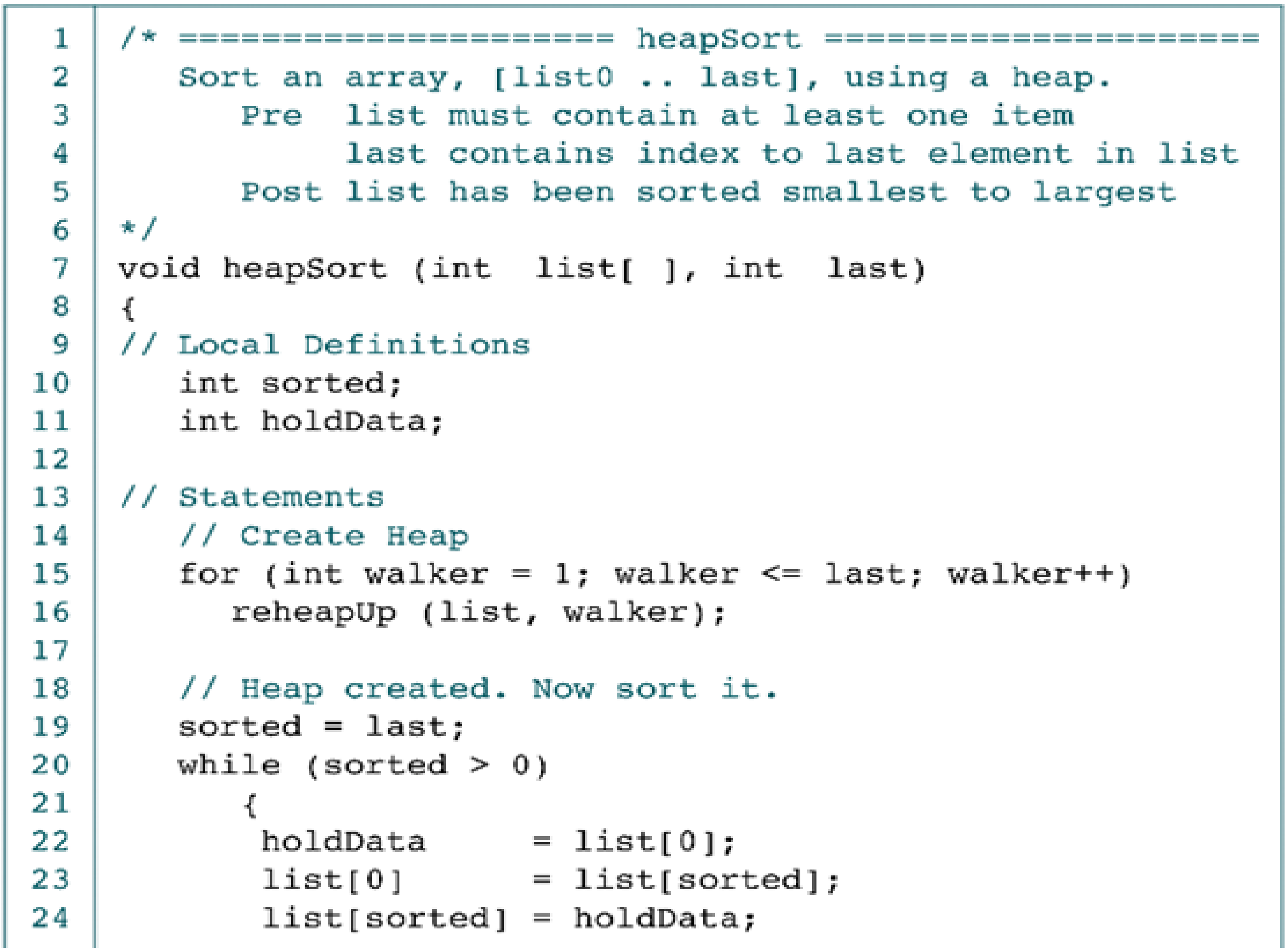
The root is the second item in the array. We skip the index zero cell of the array for the convenience of implementation. Consider i-th element of the array, then

its left child is located at 2*i index

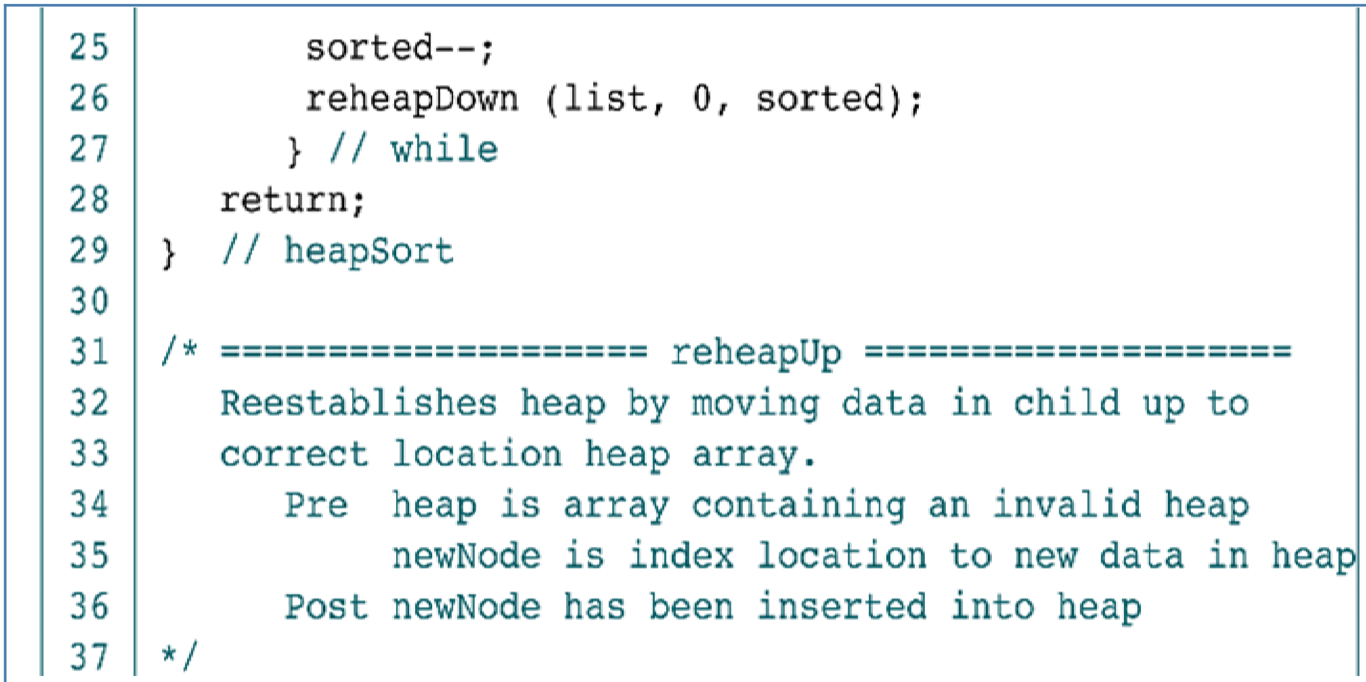
its right child is located at 2*i+1. index

its parent is located at i/2 index

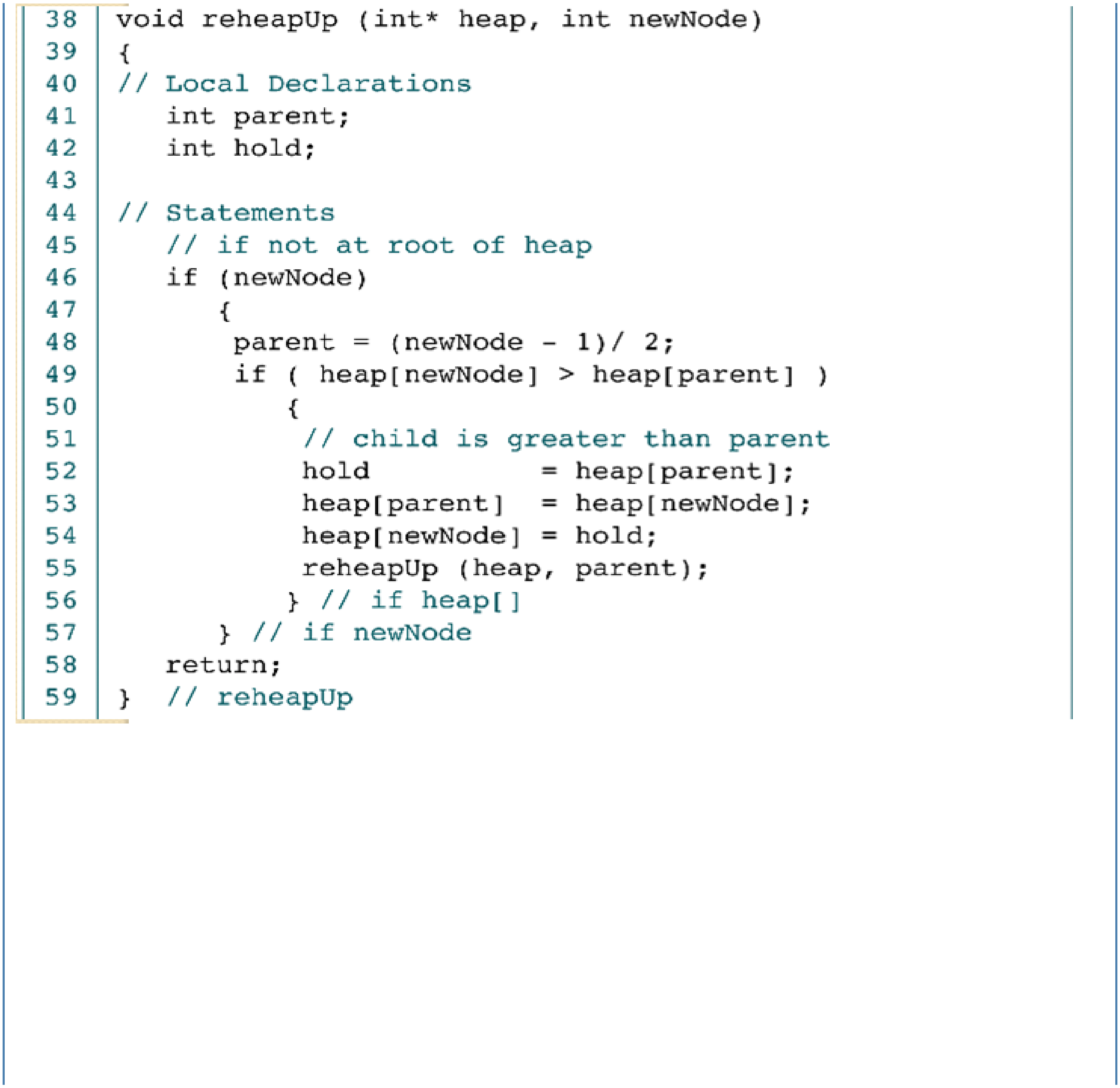
Algorithm:



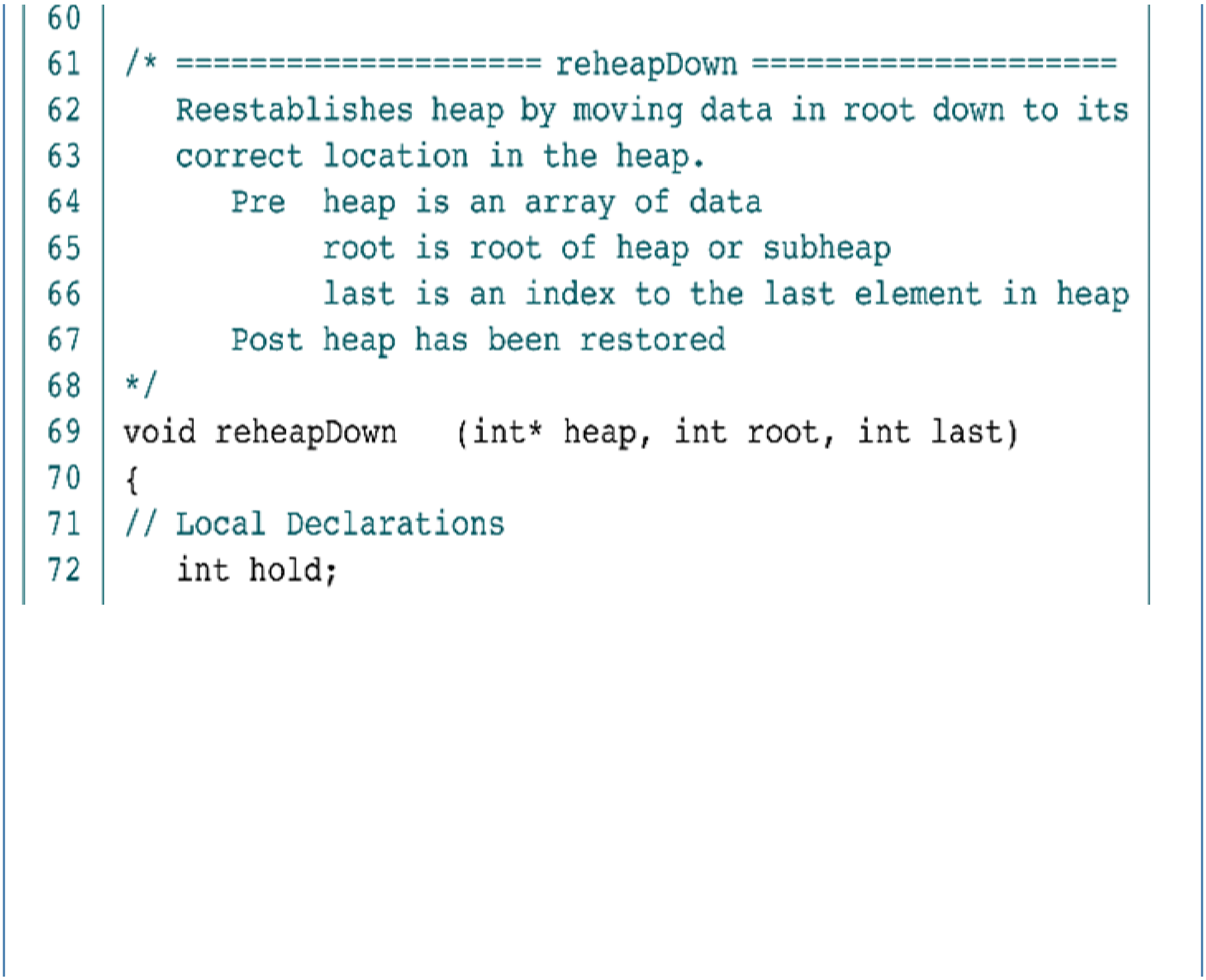
```c
/* ===================== heapSort ======================
   Sort an array, [list0 .. last], using a heap.
      Pre  list must contain at least one item
           last contains index to last element in list
      Post list has been sorted smallest to largest
*/
void heapSort (int  list[ ], int  last)
{
// Local Definitions
   int sorted;
   int holdData;

// Statements
   // Create Heap
   for (int walker = 1; walker <= last; walker++)
      reheapUp (list, walker);

   // Heap created. Now sort it.
   sorted = last;
   while (sorted > 0)
      {
       holdData     = list[0];
       list[0]      = list[sorted];
       list[sorted] = holdData;
```



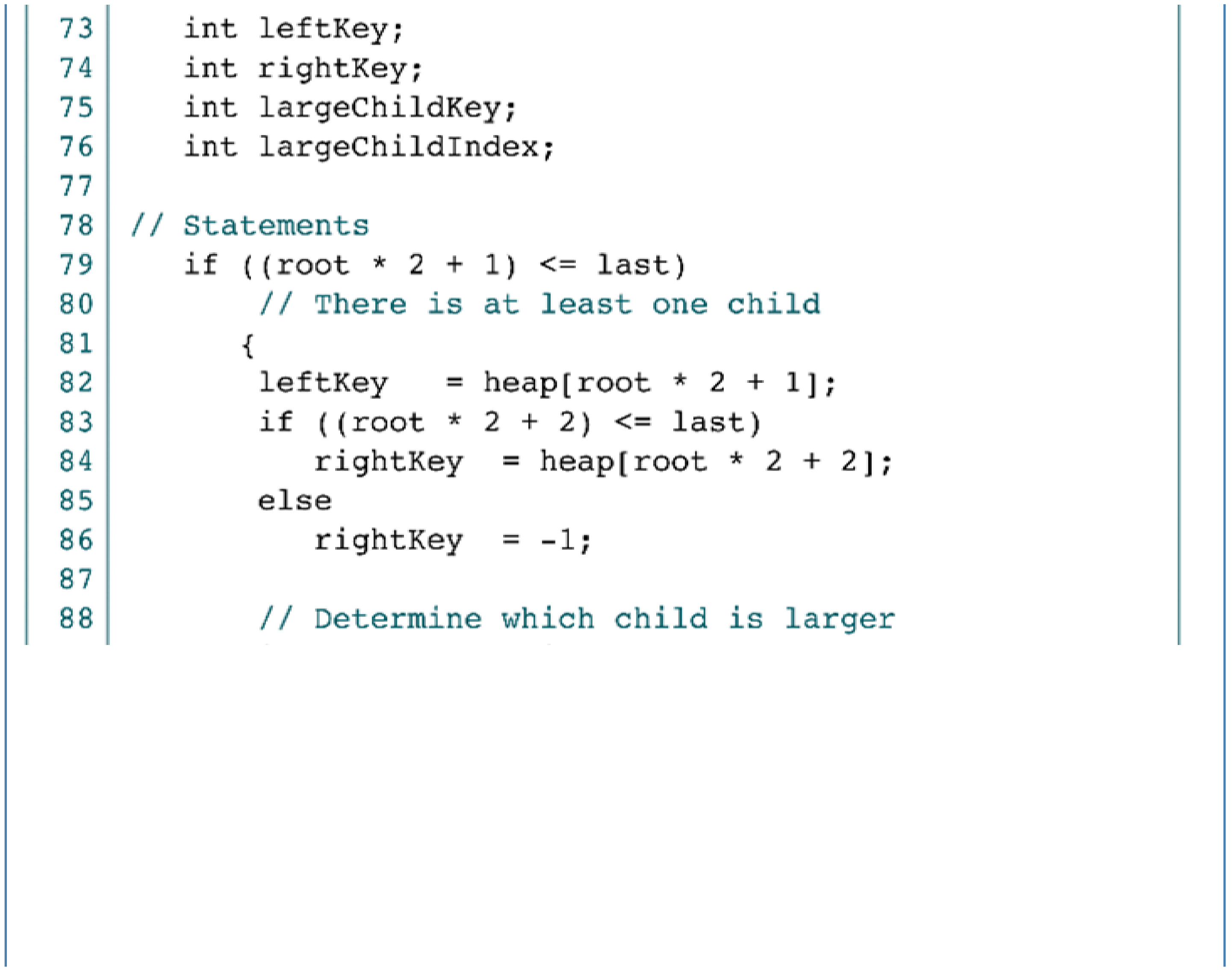
```
       sorted--;
       reheapDown (list, 0, sorted);
      } // while
   return;
}  // heapSort

/* ==================== reheapUp ====================
   Reestablishes heap by moving data in child up to
   correct location heap array.
      Pre  heap is array containing an invalid heap
           newNode is index location to new data in heap
      Post newNode has been inserted into heap
*/
```



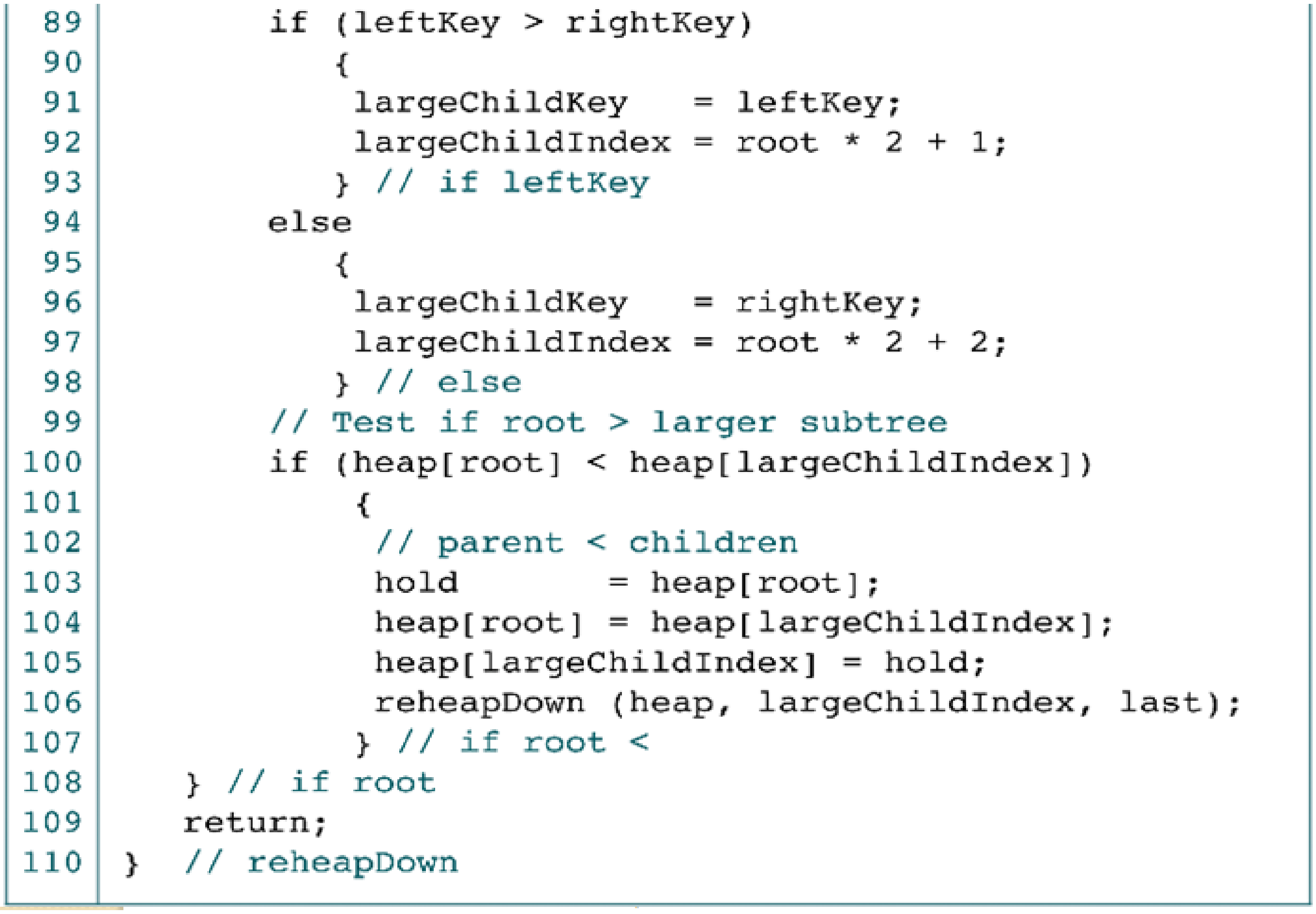
```
void reheapUp (int* heap, int newNode)
{
// Local Declarations
   int parent;
   int hold;

// Statements
   // if not at root of heap
   if (newNode)
      {
       parent = (newNode - 1)/ 2;
       if ( heap[newNode] > heap[parent] )
          {
           // child is greater than parent
           hold            = heap[parent];
           heap[parent]    = heap[newNode];
           heap[newNode] = hold;
           reheapUp (heap, parent);
          } // if heap[]
      } // if newNode
   return;
}  // reheapUp
```



```
/* ==================== reheapDown ====================
   Reestablishes heap by moving data in root down to its
   correct location in the heap.
      Pre  heap is an array of data
           root is root of heap or subheap
           last is an index to the last element in heap
      Post heap has been restored
*/
void reheapDown   (int* heap, int root, int last)
{
// Local Declarations
   int hold;
```



```
   int leftKey;
   int rightKey;
   int largeChildKey;
   int largeChildIndex;

// Statements
   if ((root * 2 + 1) <= last)
       // There is at least one child
      {
       leftKey   = heap[root * 2 + 1];
       if ((root * 2 + 2) <= last)
          rightKey  = heap[root * 2 + 2];
       else
          rightKey  = -1;

       // Determine which child is larger
```



```
       if (leftKey > rightKey)
          {
           largeChildKey   = leftKey;
           largeChildIndex = root * 2 + 1;
          } // if leftKey
       else
          {
           largeChildKey   = rightKey;
           largeChildIndex = root * 2 + 2;
          } // else
       // Test if root > larger subtree
       if (heap[root] < heap[largeChildIndex])
           {
            // parent < children
            hold       = heap[root];
            heap[root] = heap[largeChildIndex];
            heap[largeChildIndex] = hold;
            reheapDown (heap, largeChildIndex, last);
           } // if root <
   } // if root
   return;
}  // reheapDown
```

ANALYSIS OF BUILD-MAX-HEAPIFY() FUNCTION

In Build-Max-heapify() function can call Max-Heapify function about n/2 times=O(n). And running time complexity of Max-Heapify is O(h) = O(logn).So, Build-Max-Heapify runs in O(n*logn).

Tighter Bound for T(BuildMaxHeap):

Cost of a call to MaxHeapify at a node depends on the height, h, of the node –> O(h).

And most of the heapification takes place at lower level. Height of node h ranges from 0 to logn. And, at most $[n/2^{h+1}]$ nodes are there at any height h. Therefore, running time complexity of Build-Max-Heap can be expressed as:

$$\begin{aligned}
\sum_{h=0}^{\lceil \log n \rceil} \frac{n}{2^{h+1}} O(h) &= O\left(n \sum_{h=0}^{\lceil \log n \rceil} \frac{h}{2^{h+1}}\right) \\
&= O\left(\frac{n}{2} \sum_{h=0}^{\lceil \log n \rceil} \frac{h}{2^{h}}\right) \\
&= O\left(n \sum_{h=0}^{\infty} \frac{h}{2^{h}}\right) \\
&= O(2n) \\
&= O(n)
\end{aligned}$$

SYMPTOTIC ANALYSIS OF HEAP SORT

Heap Sort first call Build-Max-Heap function which takes O(n) time then it call Max-Heapify function n time, where Max-Heapify function takes O(logn) time so, total time complexity of heap-sort comes out to be O(n+n*logn) which further evaluated to O(n*logn). Summarising all this –

Time Complexity of Max-Heapify: O(logn)

Time Complexity of Build-Max-Heapify:O(n)

Time Complexity of Heap Sort-

Worst Case : O (n*logn)

Average Case : O(n*logn)

Best Case : O(n*logn)

Sorting In Place: Yes

Stable: No

Input:

Online examination of particular subject marks are given as a input (Ascending, Descending or Random order) to heap data structure.

For example following marks 90, 25, 74, 64, 80, 39, 89, 45, 59, and 49 are given as a input to heap data structure

Output:

Maximum marks obtained in that subject : 90

Minimum marks obtained in that subject : 25

Conclusion:

We have successfully completed to find the maximum and minimum marks of a given subject using heap data structure by taking the real time example such as online examination scheme.

Frequently Asked Questions (FAQ):

5. Define the terms with the help of examples: Heap Data Structure, Heapify, Max Heap, Min Heap, reHeapUp, reHeapDown.
6. Construct Max Heap for following elements 90, 25, 74, 64, 80, 39, 89, 45, 59, and 49
7. Display following elements in ascending order using heap data structure 90, 25, 74, 64, 80, 39, 89, 45, 59, and 49
8. Explain with the help of example
    - I. How to find parent node for given child.
    - II. How to find left and right child for given parent node.

ASSIGNMENT NUMBER: 11

Group F-23	**Date:**

Title: Department maintains a student information. The file contains roll number, name, division and address. Allow user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to main the data.

Continuous Assessment:	**C**	**C**	**R**	**P**	**U**	**Total**

Sign:	**Date:**	**Remark:**

ASSIGNMENT NUMBER: 11

Title: Department maintains a student information. The file contains roll number, name, division and address. Allow user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to maintain the data.

Objective:

- Implement various operations of a Sequential file, such as add, delete information of student and displaying particular information, if not exist an appropriate message is displayed etc.
- Give that advantages and disadvantages of a sequential file.
- Identify applications where a sequential file will be useful.

Environment: Open Source C++ Programming tool like G++

Theory:

Files

• Used for permanent storage of large quantity of data

• Generally kept on secondary storage device, such as a disk, so that the data stays even when the computer is shut off Data hierarchy

• Bit

– Binary digit

– true or false

– quarks of computers

- Byte
  - – Character (including decimal digits)
  - – Atoms
  - – Smallest addressable unit (difficult to get by itself)
  - – Generally, eight bits to a byte though there used to be 6-bit bytes in the 60s
- Word
  - – Collections of bytes
  - – Molecules
  - – Smallest unit fetched to/from memory at any time
  - – Number of bits in word is generally used as a measure of the machine’s addressability (32-bit machine)
- Field
  - – Collection of bytes or even words
  - – Exemplified by the name of an employee (30 characters/bytes, or 8 words on a 32-bit

machine)

- Record
  - Collection of fields
  - struct in C

- File
  - Collection of records
  - Each record in the file identified with a unique [set of] field[s], called key
  - For example student name or roll number as a key to keep the file of grades
  - The payroll of a large company may use the social security number as the key

  - Sequential file
    - Records follow one after the other
  - Random access file
    - The location of a record is a function of the key
    - Mostly used in databases
  - Indexed sequential file
    - The location of a record is dependent on an index kept in a separate file Files and streams

Sequential file

When writing a program to process sequential files, the first step is that the include directive for processing sequential files is written into the directives section of the program. This directive is:

```
#include <fstream>
```

Then next step is to create an object with a name to refer to the object that will reflect what is the object's function. Will this object be used as Input or Output.

```
ifstream objectname;    //create an input file object, name objectname assigned to object
ofstream objectname; //create an output file object, name objectname assigned to object
```

The keyword ifstream indicates an input file and the objectname is any name you wish to give to the object. It should follow the rules for creating variable names. Similarly the keyword ofstream indicates an output file.

Next step is to open the file for use by the program. The open process determines how the file will be opened, for input, for output or for appending records as well as the location and name of the file.

To open a file use:

```
        objectname.open(filename,[mode]);
```

objectname = name of object created in ifstream or ofstream

filename = d:/path/filename.ext the name of the file and if the file is located or is to be created in a directory other than the current directory, you must give the DOS path for the direction of the file.

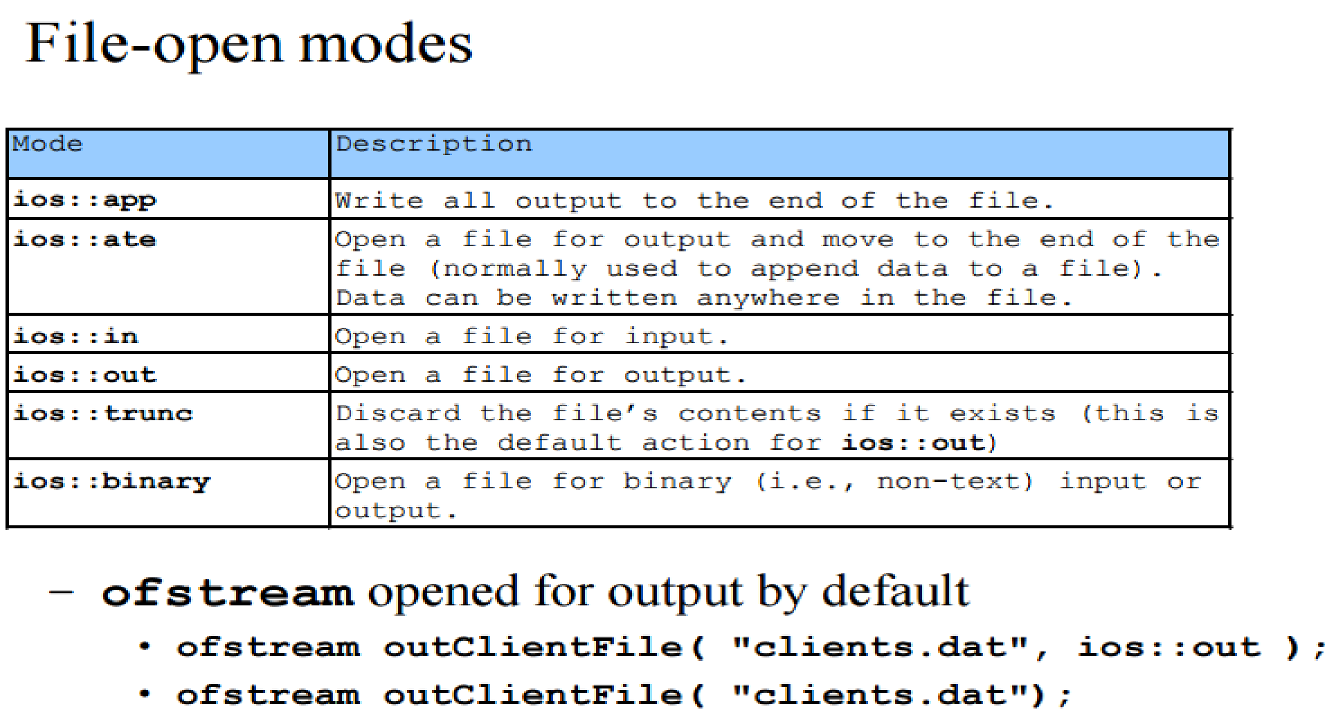
mode may be,   ios::in   input – contains data for processing by the program

ios::app append – add records to an existing file or if the file does not exist it will act

exactly like ios::out and create a new file by that name in the directory indicated.

ios::out output – create a new file. or if the file already exists you will overwrite any

existing data.



File-open modes

Mode	Description
ios::app	Write all output to the end of the file.
ios::ate	Open a file for output and move to the end of the file (normally used to append data to a file). Data can be written anywhere in the file.
ios::in	Open a file for input.
ios::out	Open a file for output.
ios::trunc	Discard the file's contents if it exists (this is also the default action for **ios::out**)
ios::binary	Open a file for binary (i.e., non-text) input or output.

- **ofstream** opened for output by default
  - **ofstream outClientFile("clients.dat", ios::out);**
  - **ofstream outClientFile("clients.dat");**

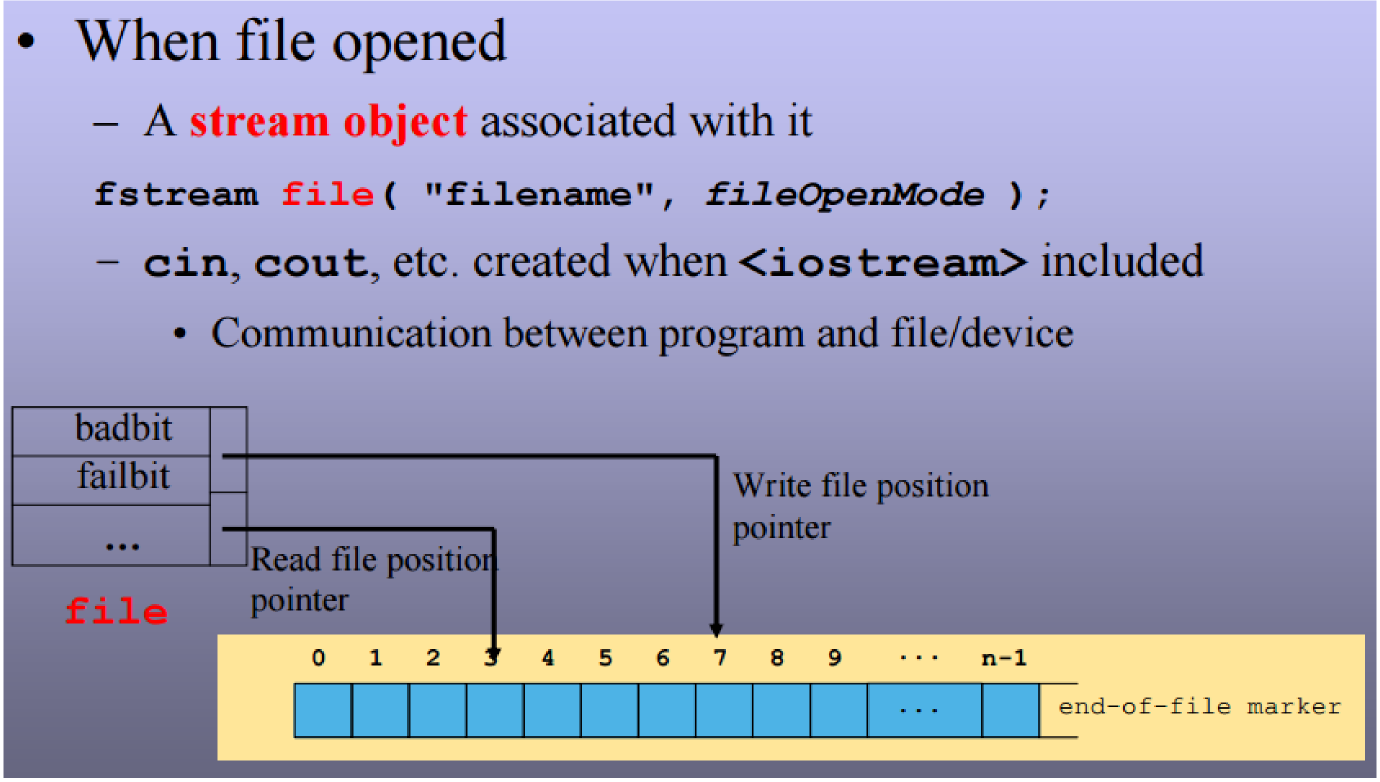
Next step is to find out if the file open was successful. If the file open was not successful need to warn the operator and take some type of corrective action or cancel the program. Then test to see if the open failed or if the open did not fail. Check applying below steps

Test if open failed:

```
if(objectname.fail() == 1)   OR
if(objectname.fail())
```

Test if open did NOT fail(open successful)

```
if(objectname.fail()==0) OR
if(!objectname.fail())
```



- When file opened
  - A **stream object** associated with it

```
fstream file( "filename", fileOpenMode );
```

  - `cin`, `cout`, etc. created when `<iostream>` included
    - Communication between program and file/device

badbit
failbit
...
file
Write file position pointer
Read file position pointer
0 1 2 3 4 5 6 7 8 9 ... n-1
...
end-of-file marker

Getting Input from a File

Reading data from a sequential file is similar to getting data from the keyboard. In place of the cin keyword, use the name of the object.

```
objectname >> data;
```

When read data from the keyboard or from a file use similar statement structure syntax. First the name of the object that will be reading the data, cin or the objectname assigned to a file. Next, the extraction operator " >> " and finally the name of the variable that will contain the input value at the end of the instruction.

```
Data from:
        cin >> variablename;                    // get data from the keyboard
        inFile >> variablename;                 // get data from a sequential file
```

Writing Data to a File

Writing data to a sequential file is similar to the cout statement. In place of the cout keyword, use the name of the object,

```
        objectname << data << endl;
```

When write to the screen (common output) or to an output file (new file or append to an existing file) we use the following syntax. First the name of the output object followed by the insertion operator " << " and finally the name of the variable containing the data to be output.

Examples:

```
        outFile << score << endl;
        payOut << wkPay << " " << hours << endl;

        outData << textline << endl;

        studentData << lname << ";" << fname << ";" << socsecnbr << endl;
```

Closing file

- outFile.close()
- Automatically closed when destructor called.

Algorithm:

Input:

- Department maintains a student information. The file contains roll number, name, division and address.

- Allow user to add the details to the existing sequential file.

Output:

- Display information of particular student.
- Search the record of student, If record of student does not exist an appropriate message is displayed.

  If it is, then the system displays the student details.
- Delete the information of student.

Conclusion:

Sequentially file operation successfully competed with add the data to the file, delete the data from the file and searching the data from the file.

Frequently Asked Questions (FAQ):

9. Define the following terms with the help of examples: File, Types of file, Various modes in file.
10. Explain the importance of files?
11. Explain the various function like adding, appending, deleting, modifying, searching, and display using sequential file.