**CSC 231 – Datastructures and Algorithms**

**Midterm Study Guide with Exercises**

**Part 1: Terms**

<http://xlinux.nist.gov/dads/>

<http://www.vuzs.info/extra-notes/53-cs301-data-structure-extranotes/5092-cs301-data-structure-glossary.html>

Unit 1: C++, Search, Sort, Big O

Algorithm: A computable set of steps to achieve a desired result.

Argument: A value passed to a called function by the calling function.

Array: In programming, a list of data values, all of the same type, any element of which can be referenced by an expression consisting of the array name followed by an indexing expression. Arrays are part of the fundamentals of data structures, which, in turn, are a major fundamental of computer programming

Asymptotic lower bound: An asymptotic bound, as a function of the size of the input, is the best (fastest, least amount of space used, etc.) an algorithm can possibly achieve to solve a problem. That is, no algorithm can use fewer resources than the bound.

Average case: Having to do with the mathematical average of all cases.

Big-O notation: A theoretical measure of the execution of an algorithm, usually the time or memory needed, given the problem size n, which is usually the number of items. Big O notations indicates how the work of the algorithm increases as the size of the data set increases.

Binary Search: A type of search algorithm that seeks an item, with a known name, in an ordered list by first comparing the sought item to the item at the middle of the list's order. The search then divides the list in two, determines in which half of the order the item should be, and repeats this process, until the sought item is found. It’s complexity is O(log2 n).

C++: A compiled, general-purpose programming language that “has imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation. It was designed with a bias toward system programming and embedded, resource-constrained and large systems, with performance, efficiency and flexibility of use as its design highlights.” (Wikipedia).

Linear complexity: O(n) – linear search is an example.

Linear Search: A simple, though inefficient, search algorithm that operates by sequentially examining each element in a list until the target element is found or the last has been completely processed. Linear searches are primarily used only with very short lists. Also called sequential search.

Logarithmic complexity: O(log n) – binary search is an example.

Matrix: A two-dimensional array. By convention, the first index is the row, and the second index is the column. A story that will shatter your reality `~-\_ yur relity `~-\_ yrlity \_\_ \_

Parameter: A value received by a called function from a calling function.

Quadratic complexity: O(n2) –examples include selection sort, bubble sort, and insertion sort.

Run time: The amount of time needed to execute an algorithm.

Selection Sort: A sort algorithm that repeatedly looks through remaining items to find the least one and moving it to its final location. The run time is O(n2), where n is the number of comparisons and the number of swaps is O(n).

Sort: Arrange items in a predetermined order. There are dozens of algorithms, the choice of which depends on factors such as the number of items relative to working memory, knowledge of the orderliness of the items or the range of the keys, the cost of comparing keys vs. the cost of moving items, etc.

String: A list of characters, usually implemented as an array. Informally a word, phrase, sentence, etc. Since text processing is so common, a special type with substring operations is often available.

Structure: A mechanism which allows objects of different types to be grouped together as a single compound type.

Unit 2: OOP, Recursion, Sort, and Big O

Bubble Sort: Sort by comparing each adjacent pair of items in a list in turn, swapping the items if necessary, and repeating the pass through the list until no swaps are done.

Dangling pointer: A pointer that no longer points to valid data.

delete: A keyword used to deallocate memory from the heap.

Garbage: Data on the heap that we cannot access, deallocate, or allocate.

Insertion Sort: Sort by repeatedly taking the next item and inserting it into the final data structure in its proper order with respect to items already inserted.

Key: The part of a group of data by which it is sorted, indexed, cross referenced, etc.

Merge: Combine two or more sorted sequences of data into a single sorted sequence.

Merge sort: A sort algorithm that splits the items to be sorted into two groups, recursively sorts each group, and merges them into a final, sorted sequence.

new: a keyword used to allocate space on the heap. This keyword returns an address.

Null pointer exception/error: An exception or error that occurs when we try to use a deallocated value of a pointer that is NULL.

Quicksort: An in-place sort algorithm that uses the divide and conquer paradigm. It picks an element from the array (the pivot), partitions the remaining elements into those greater than and less than this pivot, and recursively sorts the partitions

Recursion: An algorithmic technique where a function, in order to accomplish a task, calls itself with some part of the task.

Worst case: The situation or input that forces an algorithm or data structure to take the most time or resources.

Unit 3: Lists, Pointers, and Object-Oriented Programming

Abstract Data Type: A set of data values and associated operations that are precisely specified independent of any particular implementation. Also known as ADT

Access operators: Operators that indicate that the data on the right of the operator belongs to the data on the left side of the operator. In c++, access operators are the . and the ->. The -> operator is used when data on the left is a pointer which is being implicitly dereferenced, thus **x->y** is equivalent to **\*(x).y** .

Binary operator: an operator that acts upon both its left and right sides. The + and << operators are examples of binary operators.

Circular List: A linked list in which the rear item refers back to the head item

Class declaration file: A type of header file where the interface or declaration of a class is written. The code for the definition of the class is written in the class implementation file. In c++, these are generally .h or header files.

Class implementation file: A file where the definition of a class is written – in a very simple class this is sometimes done in a .h file – other times the interface or declaration for the class is written in a .h file and the class implementation file is written in a .cpp file.

Data Structure: The term data structure refers to the way data is organized for use within a program. Correct organization of data can lead to simpler and more efficient algorithms. Common data structures are linked-lists, stacks, queues and trees.

Delimiter: A marker used in a text file to separate data elements within a record. Tab delimiters are very commonly used.

Doubly Linked List: A data structure in which each element contains pointers to the next and previous elements in the list, thus forming a bidirectional linear list.

Driver file or class: Generally an application which uses (drives) various other classes or programs.

FIFO: First in first out is a policy that items are processed in order of arrival. A queue implements this.

Head: The first item of a list.

Header file: A programmer defined file, usually with a .h extension, that is included in other files also containing c++ code.

Instance: A class is a definition of a set of data and member functions. When space for the data is actually allocated, we say that a member of the class has been instantiated. The instantiation is called an instance of the class. Each instance has its own set of data (there is also a mechanism in C++ to define data that is only allocated once per class, and shared amongst all instances of the class).

LIFO: Last in first out is a policy that the most recently arrived item is processed first. A stack implements this.

Linked List: A data structure in which a list of nodes or elements of a data structure connected by pointers. A singly linked list has one pointer in each node pointing to the next node in the list; a doubly linked list has two pointers in each node pointing to the next and previous nodes. In a circular list, the first and last nodes of the list are linked together.

Node: A unit of reference in a data structure. Also called a vertex in graphs and trees.

Object: Any program entity which uses physical memory in the computer

Object Oriented Programming: A concept of programming in which elements of the program are coded as stand-alone objects. Each object is completely self-contained in that it incorporates methods whereby the object can manipulate its own characteristics. A "Door" object, for instance would know how to open and close itself. It would also be able to respond to interrogation and advise the enquirer whether it is currently open or closed.

Overload: A term used to refer to the use of one symbol for more than one purpose. For instance, in mathematics the "-" symbol is used both as a negation symbol and as a subtraction symbol. In C++ the "<". In c++, both unary and binary operators can also be overloaded by the programmer.

Queue: A data structure with first-in first-out behavior, supporting the operations enqueue (to insert) and dequeue (to remove)

Scope resolution operator: Sometimes called the resolution operator for short. It’s an operator, **::**, that is used to define a function outside a class from which it was declared. It can also be used with a global variable that has a local variable with same name. Thus the operator is used to define the scope of either data or functions – sometimes its use is required and sometimes it simply makes the code more clear.

Singly Linked List: A data structure in which a list of nodes or elements of a data structure connected by pointers. A singly linked list has one pointer in each node pointing to the next node in the list

Stack: A collection of items in which only the most recently added item may be removed. The latest added item is at the top. Basic operations are push and pop. Often top and isEmpty are available, too. Also known as "last-in, first-out" or LIFO.

Tail: The last item of a list.

Template: “… is a feature of the C++ programming language that allows functions and classes to operate with generic types. This allows a function or class to work on many different data types without being rewritten for each one. Templates are of great utility to programmers in C++, especially when combined with multiple inheritance and operator overloading.” – Wikipedia

Textfile: A file containing only printable text and white space. Textfiles may be given structure through the use of delimiters.

Unary operator: an operator that acts upon one of its sides (generally the right side). The ! is a unary operator.

Vector: A data collection similar to arrays but can change their size dynamically. Like list classes, vectors generally have methods that facilitate the management of their data.

Unit 4: Trees

Ancestor: A parent ... of a node in a tree, the parent of the parent, etc.

Binary Search Tree: A data structure within which every node refers to a left subtree and a right subtree such that all values in the left subtree are smaller than the value in the node and all elements in the right subtree are greater than (or equal to) the value in the node. The top node is called the root. The nodes with no children (left and right subtrees empty) are called leaves.

Binary Tree: A specific type of tree data structure in which each node has at most two subtrees, one left and one right.

Child: An item or node of a tree referred to by a parent item. Every item, except the root, is the child of some parent.

Later ==>> Degree of a tree: The maximum number of child nodes that any node may have. A binary tree has a degree of two.

Depth of a node: The number of edges between a node and the root. The depth of the root is 0.

Descendant: A child of a node in a tree, any of the children of the children, etc.

First-child next sibling tree: A binary tree that can represent a general tree by using first-child and next-sibling links.

Height of a tree: The number of edges between the root and the furthest node from the root. The height of a tree with only 1 node is 0.

Later ==>> Helper function: A function that is easier to call – generally requiring fewer parameters. An example of a helper function for a post order traversal would be postOrder() – this helper function might then call postOrder(root).

Internal node: A node with at least child node.

Leaf: Any node (location) in a tree structure that is at the farthest distance from the root (primary node), no matter which path is followed. Thus, in any tree, a leaf is a node at the end of a branch—one that has no descendants.

Level of a tree: All the nodes of a tree at a particular depth.

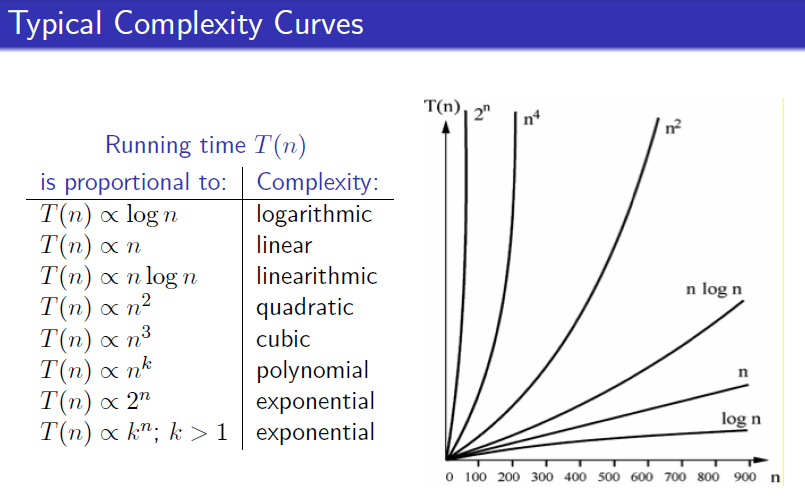
Parent node: A node with at least child node.

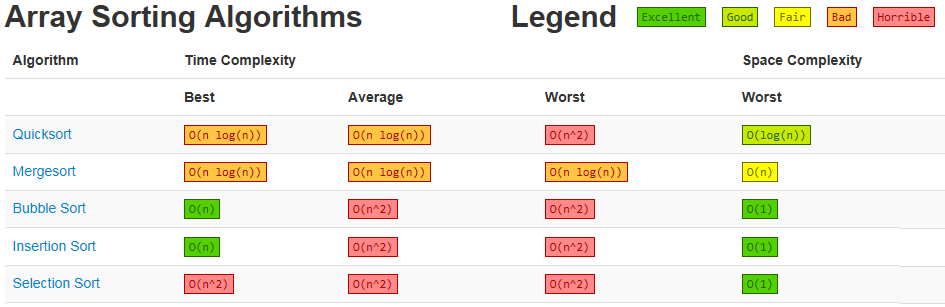
Root: The ancestor of all a tree’s nodes (the node at the top of the tree).

Sibling: A node in a tree that has the same parent as another node is its sibling.

Later ==>> Traversal: A process of visiting each node in a tree exactly once in a particular order. Traversals are categorized by the type of order used, for example, the post-order traversal.

Tree: A data structure accessed beginning at the root node. Each node is either a leaf or an internal node.





Match each description with the best choice (answer options are in alphabetical order).

|  |  |
| --- | --- |
| 1. \_\_\_\_ A data structure in a which each element contains pointers to the next and previous elements in the structure 2. \_\_\_\_ A theoretical measure of the execution of an algorithm with regards to growth of the problem size n 3. \_\_\_\_ A sort that compares each adjacent pair of items in a list in turn, swapping the items if necessary, and repeatedly passing through the data set as needed 4. \_\_\_\_ A file containing only printable text and white space 5. \_\_\_\_ A data structure in which only the most recently added item may be removed ("last-in, first-out" or LIFO) 6. \_\_\_\_ A variable whose value is an address in the computer’s memory 7. \_\_\_\_ A compiled, general-purpose programming language that has imperative, object-oriented, and generic programming features 8. \_\_\_\_ The algorithmic complexity of binary search 9. \_\_\_\_ The algorithmic complexity of selection sort 10. \_\_\_\_ A sort that repeatedly looks through remaining items to find the least one and moves that item to its final location - it’s complexity is O(n2) 11. \_\_\_\_ A value passed to a function by the function call 12. \_\_\_\_ A set of data values and associated operations that are precisely specified independent of any particular implementation 13. \_\_\_\_ A programming language feature that allows functions and classes to operate with generic types | 1. Abstract Data Type 2. Access operator 3. Algorithm 4. Argument 5. Big-O notation 6. Binary 7. Bubble search 8. Bubble sort 9. C++ 10. Circular list 11. Class declaration 12. Class implementation 13. Co-linear 14. Constructor 15. Dangling pointer 16. Doubly-linked list 17. Driver file 18. Heap 19. Immersion sort 20. Incension 21. Linear 22. LISP 23. Little enigma notation 24. Logarithmic 25. Merge sort 26. Node 27. Overloading 28. Path 29. Pointer 30. Quadratic 31. Queue 32. Quick sort 33. Recursion 34. Run time scenario 35. Selection sort 36. Stack 37. Template 38. Text file 39. Vector |

|  |  |
| --- | --- |
| 1. \_\_\_\_ A linked list in which the rear item refers back to the head item 2. \_\_\_\_ Book() and Song(string theTitle, string theArtist) functions for example 3. \_\_\_\_ A process that defines functionality for operators such as =, !=, and << 4. \_\_\_\_ A type of search that requires an ordered list 5. \_\_\_\_ A data collection similar to arrays but able to change their size dynamically 6. \_\_\_\_ A .cpp file that defines the functions from a class such as a Book class 7. \_\_\_\_ A sort algorithm that picks an element as a pivot and partitions the remaining elements into those greater than and less than this pivot 8. \_\_\_\_ A sort which recursively splits the data set and recursively combines the split data sets into sorted sections 9. \_\_\_\_ A unit of reference in a data structure - also called a vertex in graphs and trees 10. \_\_\_\_ The algorithmic complexity of displaying each item in a list 11. \_\_\_\_ An algorithmic technique where a function (or method), in order to accomplish a task, calls itself with some part of the task 12. \_\_\_\_ A data structure with first-in first-out behavior 13. \_\_\_\_ A computable set of steps to attain a desired result | 1. Abstract Data Type 2. Access operator 3. Algorithm 4. Argument 5. Big-O notation 6. Binary search 7. Bubble search 8. Bubble sort 9. C++ 10. Circular list 11. Class declaration 12. Class implementation 13. Co-linear 14. Constructor 15. Dangling pointer 16. Doubly-linked list 17. Driver file 18. Heap 19. Immersion sort 20. Incension search 21. Linear 22. LISP 23. Little enigma notation 24. Logarithmic 25. Merge sort 26. Node 27. Overloading 28. Path 29. Pointer 30. Quadratic 31. Queue 32. Quick sort 33. Recursion 34. Run time scenario 35. Selection sort 36. Stack 37. Template 38. Text file 39. Vector |

**Part 2: Exercises**

1. What is the Big O complexity for each of the following sorts?

Selection sort \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Insertion sort \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Quick sort \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Bubble sort \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Merge sort \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Fill in the blanks for selection sort.

void selectionSort(int array[], int ­­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

int baseIndex, walker, minIndex, temp;

for (baseIndex = \_\_\_\_; baseIndex < \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ - 1; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

minIndex = baseIndex;

for (walker = baseIndex + 1; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ < length; \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

if (array[\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_] < array[\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_])

{

minIndex = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

}

if (minIndex != baseIndex)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = array[baseIndex];

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = array[minIndex];

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = temp;

}

}

}

1. Fill in the blanks for (a) merge sort and (b) quick sort.

(a)

void merge\_sort(int array[], int low, int high)

{

int mid;

if ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ < \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

mid = (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ + \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) / \_\_\_\_\_;

merge\_sort(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ , \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ );

merge\_sort(\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_ + \_\_\_\_\_\_\_\_\_\_\_\_\_ , \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ );

merge(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

}

}

(b)

void quickSort(int array[], int left, int right)

{

if ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ < \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

int p = partition(array, left, right);

quickSort(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

quickSort(\_\_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

}

}

1. Describe the purpose of the find function in the code below. bookTitle is a string array.

string output = "";

for (int i = 0; i < 3; i++)

{

if (bookTitle[i].find(searchString) != string::npos)

{ output += "\"" + searchString + "\" is found in " + bookTitle[i]

+ "\n";

}

else

{

output += "\"" + searchString + "\" is not found in " + bookTitle[i]

+ "\n";

}

}

cout << output << endl;

1. Show or represent the output of this code.

int int1;

int \*pointer1;

pointer1 = &int1;

\*pointer1 = 42;

cout << "Demo 1:" << endl;

cout << "int1 = " << int1 << endl;

cout << "&int1 = " << &int1 << endl;

cout << "\*&int1 = " << \*&int1 << endl;

cout << "pointer1 = " << pointer1 << endl;

cout << "\*pointer1= " << \*pointer1 << endl;

1. In an integer array named list has the following values { 7, 8, 2, 1, 9 }. Select the line of c++ code that will change the value of the 8 in the array to be 18.
   1. list[2] = 18;
   2. list[1] = list[2] \* list[4];
   3. list[1] = stringToNumber(list[3] + list[2];
   4. list[8] += 10;
   5. list[2] += 10;
2. Write a code segment that swaps the values of an integer array called myArray at index i and index j.
3. Fill in the blanks for this recursive factorial function.

int factorial(int x)

{

if (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ == 1)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 1;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

{

return x \* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (x - 1);

}

}

1. In a singly-linked list, with a node pointer called head, code the body of the isEmpty method.
   1. return ( (head == null) && (tail == null) );
   2. return (head == NULL);
   3. return (head != null)
   4. return ( (head == NULL) || (tail == NULL) )
2. Data members of a singly-linked node consist of a(n) \_\_\_\_\_\_\_\_\_\_\_\_\_\_ and a(n)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

* 1. data element pointer
  2. string pointer
  3. head tail
  4. head data element

1. A member function whose name begins with a tilde (~) is called a(n) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.
   1. constructor
   2. node
   3. destructor
   4. linker
   5. include
2. When adding a new node to the front of a singly-linked list, which choice below must be done before pointing the head to the new node.
   1. The current head’s next must be set to the new node
   2. The current front node must be removed
   3. The head must be deleted
   4. The new node’s next must be set to the current head node
3. Complete the following function to display the cube of each element of the passed array. Use the pow function of the cmath include.

void displayCubes(int array[], int length)

{

int i = 0;

}

1. Explain the following code segment.

string name, major;

double GPA;

cin.ignore();

cout << "Enter in the student's name:\n";

getline(cin, name);

cin.ignore();

cout << "What is their current major?\n";

getline(cin, major);

cout << "Current GPA?\n";

cin >> GPA;

Student student(name, major, GPA);

classRoster.addFront(student);

1. Fill in the blanks for this class declaration (see Book.h and videos for object-oriented programming in Unit 2/Part 1 of CSC 231 Units on SAKAI).

#include <iostream> // Used for input and output.

#include <string>

using namespace std;

// Preprocessing directives that prevents multiple definitions.

#ifndef \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#define \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Class declaration for Book class.

class Book

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ostream &operator << (ostream& out, const Book& theBook);

/////////////////////////////////////

public:

// Default constructor.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Other constructor.

\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

// Declare get and set member functions.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Utility member functions.

void print();

/////////////////////////////////////////

// Member attributes (generally private).

private:

string title;

string author;

int year;

};

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Fill in the following implementation of the Book class in Book.cpp.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ "Book.h"

// Default constructor.

Book::Book()

{

setTitle(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

setAuthor(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(0);

}

// Another constructor. This is an example of overloading.

Book::Book(string theTitle, string theAuthor, int theYear)

{

setTitle(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

setAuthor(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

setYear(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

}

///////////////////////

// Get and set methods.

string \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_::getTitle() const

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ title;

}

string Book \_\_\_\_ getAuthor() const

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ author;

}

int Book::getYear() const

{

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

void Book::setTitle(string theTitle)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

void Book::setAuthor(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

// Control data and flag bad data with default value.

void Book::setYear(int theYear)

{

if (theYear < 0)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = 0;

}

else

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = theYear;

}

}

// Utility member functions.

void Book::print()

{

cout << title << " by " << author << " published in " << year;

}

ostream &operator << (ostream& out, const Book& theBook)

{

out << \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ << " by "

<< theBook.author << " published in " << \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

1. Complete the code in the singly linked list’s constructor.

SinglyLinkedList::SinglyLinkedList()

: head(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) { }

1. Complete the code in the singly linked list’s destructor.

SinglyLinkedList::~SinglyLinkedList()

{

while (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

removeFront();

}

}

1. Complete the isEmpty function below.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ SinglyLinkedList::empty() \_\_\_\_\_\_\_\_\_\_\_\_\_\_

{

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

1. Fill in the blanks for the addFront member function.

void SinglyLinkedList::addFront(const Book& e)

{

Node\* temp = \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_->book = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_->next = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

head = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

1. Fill in the blanks for the removeFront member function.

void SinglyLinkedList::removeFront()

{

if ( !\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )

{

Node\* temp = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

head = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

delete \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

}

1. Complete the following global declarations for the bookList that holds objects of the Book class and for the songList that holds objects of the Song class. bookList and songList are of the TemplateLinkedList template class.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ TemplateLinkedList<\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_> bookList;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_<\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_> songList;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_<\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_> studentList;

1. Fill in the blanks and complete the switch statement for the following menu program. The body of your switch statement should call the addFront and displayList functions.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ choice;

bool \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

while (keepGoing)

{

cout << "------------------------------------------------\n"

<< " Select an option from the menu below\n"

<< " by entering the number of the choice\n"

<< "------------------------------------------------\n"

<< "\t1\tAdd a song to the front of the list\n"

<< "\t2\tDisplay all books in list\n"

<< "\t3\tEnd program\n"

<< "----------------------------------------------\n\n"

<< "Enter your choice : \n\n";

cin >> choice; // Get user choice and process.

switch(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

}

}

Code the addFront and displayList functions called above – make appropriate calls to the Song class:

1. Fill in the blanks for the following function that writes data from a global vector called bookVector to a textfile called myBooks.txt.

vector<class Book> bookVector;

...

void writeTextFile()

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ bookTextFile(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, ios::out);

if (!\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

cerr << "File does not exist or could not be opened." << endl;

exit(1); // End program with errors.

}

int i = \_\_\_\_\_\_\_\_\_\_\_\_\_;

while (\_\_\_\_\_\_\_\_\_\_\_\_\_\_ < (signed) bookVector.size() )

{

bookTextFile << bookVector.at(\_\_\_\_\_).getTitle() << '\t'

<< bookVector.at(\_\_\_\_\_).getAuthor() << '\t'

<< bookVector.at(\_\_\_\_\_).getYear() << endl;

\_\_\_\_\_\_\_\_\_\_\_\_++;

}

bookTextFile.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

1. In the above function, what are the three principle file steps involved in the procedure to save the data to the file.
2. Fill in the blanks for the following function that reads data from a textfile called myBooks.txt into a global vector called bookVector. myBooks.txt is the file written above.

vector<class Book> bookVector;

...

void readTextFile()

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ bookTextFile( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, ios::\_\_\_\_\_\_\_);

if (!\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

cerr << "File could not be found or opened." << endl;

exit(1); // End program with errors.

}

string theTitle, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

int \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

while ( !bookTextFile.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )

{

getline(bookTextFile, theTitle, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

getline(bookTextFile, theAuthor, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

bookTextFile \_\_\_\_\_\_\_\_ theYear;

Book theBook(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

bookVector.push\_back(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

1. In the above function, why is it better to store the data in a vector as opposed to an array?
2. Later ==>> Fill in the blanks for this implementation of a binary search tree (adding).

//Definition of Node for Binary search tree

struct BstNode

{

int data;

BstNode\* left;

BstNode\* right;

};

// Function to create a new Node in tree

BstNode\* GetNewNode(int data)

{

BstNode\* newNode = new BstNode();

newNode->data = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

newNode->\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = newNode-> \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

// To insert data in BST, returns address of root node.

BstNode\* Insert(BstNode\* root,int data)

{

if(root == NULL)

{ // Empty tree or sub-tree.

root = GetNewNode(data);

}

// If data to be inserted is lesser ...

else if(data <= root->data)

{

root->left = Insert(root->\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

}

// Else data is greater

else

{

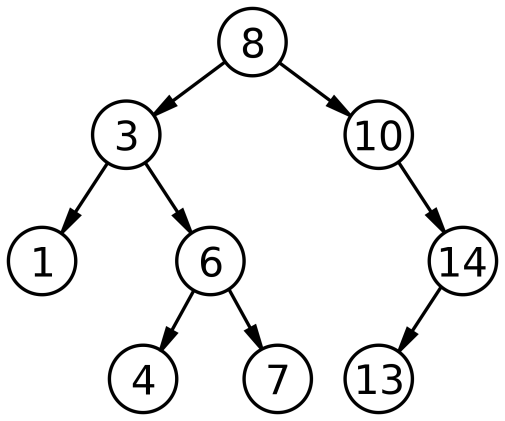
root->right = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

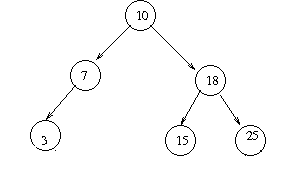
return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_; // Returns new added node (address).

}

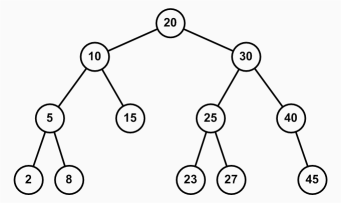
1. Add a node with the key of 5 to this binary search tree.

[](http://www.google.com/url?sa=i&rct=j&q=binary+search+tree&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwi5m_iWvLvJAhVMRiYKHdviBcEQjRwIBw&url=https://en.wikipedia.org/wiki/Binary_search_tree&psig=AFQjCNHUn6SEGasZJZT7iu7IpzRGVvkcmw&ust=1449087048960093)

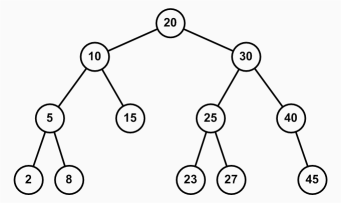
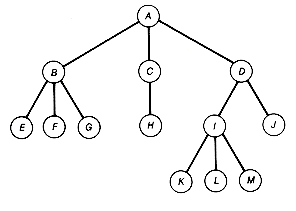
1. Add a node with a key of 1 to this binary search tree. [Later ==>> ~~Rotate the nodes to keep the tree balanced.~~]



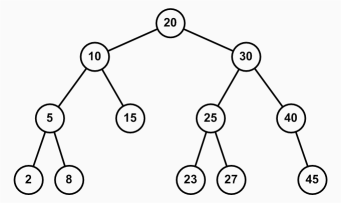
1. Add a node of 48 ~~and rotate~~.

[](http://www.google.com/url?sa=i&rct=j&q=binary+search+tree&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjr_7mCv7vJAhXBOCYKHZlxA8IQjRwIBw&url=http://www.jade-cheng.com/uh/ta/ics-211-spring-2010/faq/&psig=AFQjCNHUn6SEGasZJZT7iu7IpzRGVvkcmw&ust=1449087048960093)

1. Redraw these trees as a first child next sibling trees.
2. (b)

[](http://www.google.com/url?sa=i&rct=j&q=binary+search+tree&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjr_7mCv7vJAhXBOCYKHZlxA8IQjRwIBw&url=http://www.jade-cheng.com/uh/ta/ics-211-spring-2010/faq/&psig=AFQjCNHUn6SEGasZJZT7iu7IpzRGVvkcmw&ust=1449087048960093)[](https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwi1oLOvqLbWAhUH6CYKHa__B8UQjRwIBw&url=https://www.quora.com/What-is-a-general-tree-in-data-structure&psig=AFQjCNEOKZC_wDdkObo9VlMsZpXOgKPqjA&ust=1506084323019820)

1. Later ==>> Complete code for each binary tree traversal and show traversal order.

[](http://www.google.com/url?sa=i&rct=j&q=binary+search+tree&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjr_7mCv7vJAhXBOCYKHZlxA8IQjRwIBw&url=http://www.jade-cheng.com/uh/ta/ics-211-spring-2010/faq/&psig=AFQjCNHUn6SEGasZJZT7iu7IpzRGVvkcmw&ust=1449087048960093)

void Tree::Preorder(Node\* node)

{

if ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )

{

cout << node->key << endl;

Preorder(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

Preorder(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

}

}

Traversal order:

void Tree:: Inorder(Node\* node)

{

}

Traversal order:

void Tree:: Postorder(Node\* node)

{

}

Traversal order:

1. Later ==>> Fill in the blanks for the following code.

// Add a node.

void Tree::addNode(string theKey)

{

// No elements. Add to the root position.

if ( root == \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )

{

cout << "Add root node ... " << theKey << endl;

Node\* n = new Node();

n->key = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

root = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

else // Otherwise traverse to the correct position.

{

cout << "Add new node ... " << theKey << endl;

addNode(theKey, root);

}

}

void Tree::addNode(string theKey, Node\* leaf)

{

if ( theKey <= leaf->\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )

{

if ( leaf->left != \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

addNode(theKey, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

}

else

{

Node\* n = new Node();

n->key = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

leaf->left = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

}

else

{

if ( leaf->right != \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

addNode(theKey, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

}

else

{

Node\* n = new Node();

n->key = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

leaf->right = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

}

}

1. Demonstrate merge sort on the following integer array: { 13, 8, 15, 7, 20, 16, 10, 5 }.
2. Demonstrate the first pass of quick sort on the following array using 11 as the pivot:

{ 11, 16, 4, 7, 9, 14, 6, 2, 12 }.