Part I: 22 Multiple choice questions (2 points each)

Answer all of the following questions. READ EACH QUESTION CAREFULLY. Fill the correct bubble on your mark-sense sheet. Each correct question is worth 2 points. Choose the one BEST answer for each question. Assume that all given C++ code is syntactically correct unless a possibility to the contrary is suggested in the question.

Remember not to devote too much time to any single question, and good luck!

1. **Suppose we need to sort a list of employee records in ascending order, using the social security number (a 9-digit number) as the key (i.e., sort the records by social security number). If we need to *guarantee* that the running time will be no worse than *n* log *n*, which sorting methods could we use?**

   A. mergesort
   B. quicksort
   C. insertion sort
   D. Either mergesort or quicksort
   E. None of these sorting algorithms guarantee a worst-case performance of n log n or better

2. **Consider the following function `f`:**

```
int f(int n)
{
      int s = 0;

      while(n > 1)
      {
            n = n/2;
            s++;
      }

      return s;
}
```

   What is the asymptotic complexity of `f` in terms of `n`? (Pick the smallest correct answer)

   A. $O(n \log n)$
   B. $O(n)$
   C. $O(\sqrt{n})$
   D. $O(\log n)$
   E. $O(n^2)$

3. The *most* important reason for including a destructor in a class is:

A. To print a message for debugging purposes
B. To store information about an object before it goes out of scope
C. To free up resources allocated by that class
D. To reset the original object's pointer to NULL
E. To make your TA happy

4. One of these code fragments calls the copy constructor for class `A`. Which one? (Assume that `doSomething` is a void function with a parameter of the appropriate type.)

A.
```
A a;
B b;
a = b;
```
B.
```
A array[20];
```
C.
```
A a;
doSomething(a);
```
D.
```
A* a;
doSomething(a)
```
E.
```
A a;
doSomething(&a);
```

5. Consider a class `List` that implements an unordered list. Suppose it has as its representation a singly linked list with a head and tail pointer (i.e., pointers to the first and last nodes in the list). Given that representation, which of the following operations could be implemented in O(1) time?

I. **Insert item at the front of the list**
II. **Insert item at the rear of the list**
III. **Delete front item from list**
IV. **Delete rear item from list**

A. I and II
B. I and III
C. I, II, and III
D. I, II, and IV
E. all of them

6. What output does this C++ program produce?

```
#include <iostream>
using namespace std;

class Bird {
public:
      virtual void noise( );
};

void Bird::noise( ) { cout << "chirp "; }

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

class Duck: public Bird {
public:
      virtual void noise( );
};

void Duck::noise( ) { cout << "quack "; }

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

class Goose: public Bird {
public:
      virtual void noise( );
};

void Goose::noise( ) { cout << "honk "; }

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

int main() {
      Bird  tweety;
      Goose ralph;
      Duck  donald;

      tweety = donald;
      tweety.noise( );
      donald.noise( );
      ralph.noise( );

      return 0;
}
```

A. chirp quack honk
B. quack quack honk
C. chirp chirp honk
D. chirp chirp chirp
E. The program won't compile because the variable types in the assignment statement aren't compatible

7. Consider a class `List` that implements an unordered list. Suppose it has as its representation a dynamically expanding (resizeable) array. Which of these operations might need to delete some dynamically allocated storage to avoid a memory leak?

I. Default Constructor
II. Copy Constructor
III. Destructor
IV. Assignment operator

A. I and II
B. II and III
C. II and IV
D. III and IV
E. II, III, and IV

8. What is the postfix representation of this expression?

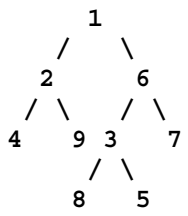
```
(12 - a) * (b + 9) / (d * 4)
```

A. `4 b * d 9 + a 12 - * /`
B. `/ 12 a - b 9 + d 4 *`
C. `12 - a * b + 9 / d * 4`
D. `12 a - b 9 + * d 4 * /`
E. None of the above

9. What is the asymptotic runtime for traversing all nodes in a binary search tree with `n` nodes and printing them in order?

A. $O(n \cdot \log(n))$
B. $O(n)$
C. $O(\sqrt{n})$
D. $O(\log(n))$
E. $O(n^2)$

10. Here is a binary tree.



If we visit the nodes of this tree using a *preorder* traversal, in what order will the nodes be visited?

A. 1 2 3 4 5 6 7 8 9
B. 1 2 4 9 6 3 8 5 7
C. 4 9 2 8 5 3 7 6 1
D. 4 2 9 1 8 3 5 6 7
E. 1 2 6 4 9 3 7 8 5

11. On your answer sheet, which test version is marked in the box to the right of your student number? (You may mark the correct version before you answer this question if you haven't done so already.)

A. A
B. B
C. C
D. D
E. E

12. Assuming that the hash function for a table works well, and the size of the hash table is reasonably large compared to the number of items in the table, the *expected* (average) time needed to find an item in a hash table containing *n* items is

A. $O(1)$
B. $O(\log n)$
C. $O(n \log n)$
D. $O(n)$
E. $O(\sqrt{n})$

13. If an internet router is receiving packets faster than it can forward them to the next location on the network, the router will

A. Return the packets to the sender, and the sender will retransmit them later
B. Store the packets locally and forward them when the next router indicates it is ready to accept more traffic
C. Discard the packets, i.e., drop them on the floor
D. The router will reboot itself, because this is never supposed to happen
E. None of the above

14. Here is a simple class and a short program that uses it. To save space, the method implementations are included in the class declaration, instead of being written separately (this is legal C++, even if not always the best style).

```
#include <iostream>
using namspace std;

class X {
private:
   int *data;  // data array

public:
   X() { data = new int[100]; }

  ~X() { delete [ ] data; }

   void set(int loc, int val) { data[loc] = val; }

   int get(int loc) { return val; }
};

int main( ) {
  X a;
  X b;
  int k;

  a.set(4,12);
  a.set(12,17);
  b = a;
  k = b.get(4);
  cout << k;
}
```

What happens when this program is executed?

A. The program executes without any errors and prints 12

B. The program won't compile, because `X` does not define assignment for objects of class X (i.e., does not contain an `operator=` method), so the assignment "`b = a;`" is not allowed here

C. The program won't compile, because `X` does not contain a copy constructor, so the assignment "`b = a;`" is not allowed here

D. The program may crash because it uses `new` and/or `delete` incorrectly

E. The program executes incorrectly because `b.get(4)` refers to an uninitialized variable, so the value assigned to `k` will be garbage

15. Members of a class can be `public`, `private`, or `protected`. What does it mean for a member of a class to be `protected`?

I. The member can be directly accessed in member functions belonging to that class
II. The member can be directly accessed in member functions belonging to classes derived from the original class
III. The member can be accessed from functions that are not members of the original or derived classes

A. I only
B. II only
C. III only
D. I and II
E. I, II, and III

16. Which of the following is *not* a key fact or rule of thumb about Computer Science?

A. $2^{10} = 1024$
B. $1 + 2 + \ldots + n = n(n+1)/2$
C. The answer is "it depends"
D. Computer Science students should get at least 8-10 hours of sleep every night, and they normally do
E. Ask not what an object can do **for** you, ask what it can do **to** itself.

17. A variable named `this` is automatically available in every member function of a class and does not need to be declared explicitly. Inside a member function for a class X, what is the type of the variable `this`?

A. `X`
B. `X*`
C. `X&`
D. `void*`
E. Something else

18. Here are a couple of related class declarations (implementations omitted).

```
class Base {
public:
   Base();
  ~Base();
   void f(int k);
   void f(double x[ ]);
   int g(double y);
};

class Extended: public Base {
   Extended();
  ~Extended();
   void f(int w);
   int g(double v[ ]);
};
```

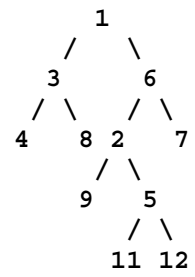
What is the relationship between the declaration of function `f` in class `Extended` and the declarations of `f` in class `Base`?

A. `Extended::f` overloads both definitions of `f` in `Base`
B. `Extended::f` overloads only `Base::f(int k)`
C. `Extended::f` overrides both definitions of `f` in `Base`
D. `Extended::f` overrides only `Base::f(int k)`
E. This program will not compile because there is only one definition of `f` in `Extended`; to fix the problem, `Extended` would need to contain both a definition of `f` with an `int` parameter and a second definition of `f` with a `double array` parameter.

19. Java supports dynamic storage allocation with `new`, but it does not provide an explicit `delete` operator like the one found in C++. Why not?

A. Java programs are small and never execute long enough to run out of memory during execution. So there is no need to reclaim unused storage until the program finishes.
B. Since Java is designed to run efficiently on small, embedded computers, the overhead required to implement `delete` is unacceptable. So Java programmers must keep track of unused storage and recycle it themselves in their code.
C. Java implementations automatically reclaim unused storage without requiring the programmer to explicitly delete it.
D. A and B
E. None of the above

20. In this binary tree,



which are the parent node(s) of the node labeled 5?

A. 2
B. 1, 6, and 2
C. 1, 6, 2, and 7
D. 1, 6, 2, 7, and 9
E. 11 and 12

21. Given the following definitions,

```
class A {
          void foo() { cout << "A::foo" << endl; }
  virtual void bar() { cout << "A::bar" << endl; }
};

class B: public A {
          void foo() { cout << "B::foo" << endl; }
  virtual void bar() { cout << "B::bar" << endl; }
};
```

what is printed when the following code is executed?

```
A * aptr = new B;
aptr->foo();
aptr->bar();
```

A. A::foo
A::bar
B. A::foo
B::bar
C. B::foo
A::bar
D. B::foo
B::bar
E. The program will not execute properly because of either a compile-time or run-time error.

22. Suppose we have the following function declarations

```
void it(int a);
void it(int *a);
void it(char a);
void it(double a);
```

Now, suppose we have the following code that includes a function call:

```
k = 10;
it(k);
```

Which of the above functions will actually be called here?

A. `void it(int a);`
B. `void it(int *a);`
C. `void it(char a);`
D. `void it(double a);`
E. The call is ambiguous because there is more than one best match, so the compiler will reject it.

Part II: 4 Programming Questions (40 points total)

Question 1. (8 points) This question involves Binary Search Trees, with nodes defined as follows:

```
struct BSTNode {
      int data;
      BSTNode* left;
      BSTNode* right;
};
```

(a) (4 points) Implement function `smallest` which returns the smallest value found in the Binary Search Tree whose root is given as the function argument. For full credit, your function should **visit only those nodes in the tree necessary** to find the smallest one, i.e., it should not visit a node if it doesn't have to. You should **assume that the tree has at least one node** (i.e., you don't have to figure out what to do if you're asked to find the smallest element in an empty tree).

[There is probably more room than necessary for the answer here. Don't panic if you don't need all the space.]

```
// iterative solution

int smallest(BSTNode* root) {
   while (root->left != NULL)
      root = root->left;
   return root->data;
}


// recursive solution

int smallest(BSTNode* root) {
   if (root->left == NULL)
      return root->data;
   else
      return smallest(root->left);
}
```

(b) (4 points) Implement the function `zero` so that it returns `true` if the binary search tree that is its argument contains the value 0. If the BST does not contain a 0, return `false`. For **full credit**, your function **should not search the entire tree if it can avoid doing so** (i.e., don't search parts of the tree that you know can't contain the value 0). Your code **should work if the tree is empty.**

```
// iterative solution

bool zero(BSTNode* root) {
   while (root != NULL)
      if (root->data == 0)
         return true;
      else if (root->data > 0)
         root = root->left;
      else // root->data < 0 here
         root = root->right;
   return false;
}

// recursive solution

bool zero(BSTNode* root) {
   if (root == NULL)
      return false;
   else if (root->data == 0)
      return true;
   else if (root->data > 0)
      return zero(root->left);
   else // root->data < 0 here
      return zero(root->right);
}
```

Question 2. (12 points) For this problem, complete the definition of function `merge` so it merges the contents of two sorted arrays into a single sorted array. The function already contains code to dynamically allocate and return a pointer to the array where the merged results should be stored.

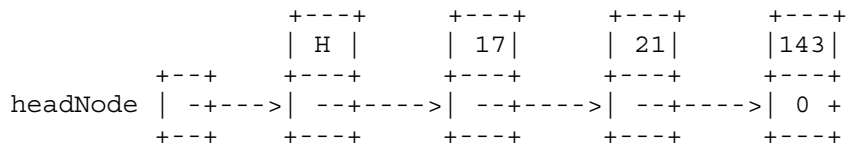
Hint: Think first and draw a picture. You may find that a counting loop of the form

for (k = 0; k < aSize+bSize; k++) { "put the right thing in ans[k]" }

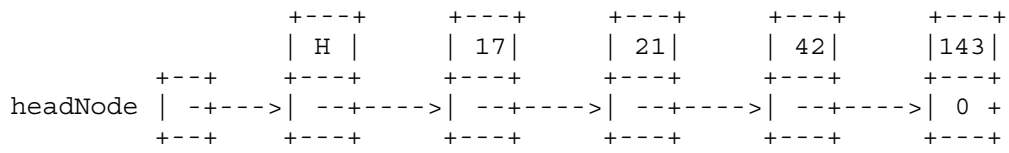
is ***not*** necessarily the easiest way to solve the problem.

```
// merge the sorted arrays a and b and return a pointer to
// a new array containing the merged result.
// precondition: a and b are sorted in ascending order, i.e.,
// a[0]<=a[1]<=...<=a[aSize-1] and b[0]<= b[1]<=...<=b[bSsize-1]
// postcondition: ans[0]<=ans[1]<=...<=ans[aSize+bSize-1]

int* merge(int a[], int aSize, int b[], int bSize) {

   int *ans = new int[aSize+bSize];  // merged result

   int k;          // ans[k] is next unfilled slot in result
   int aPos,bPos; // a[aPos] and b[bPos] are first items in a and b
                  //   not yet copied to ans

   // merge arrays until one source array has been copied completely
   k = aPos = bPos = 0;
   while (aPos < aSize && bPos < bSize) {
      if (a[aPos] <= b[bPos]) {
         ans[k] = a[aPos];
         aPos++;
      } else {
         ans[k] = b[bPos];
         bPos++;
      }
      k++;
   }

   // copy remainder of either a or b, whichever has remaining items
   while (aPos < aSize) {
      ans[k] = a[aPos];
      k++; aPos++;
   }

   while (bPos < bSize) {
      ans[k] = b[bPos];
      k++; bPos++;
   }

   return ans;
}
```

Question 3. (14 points) Suppose we use a singly linked list with a header node to store a ***sorted*** list of integers. For example, if the numbers in the list are 21, 17, and 143, the list should look like this :



Notice that variable `headNode` points to a header node that does not contain list data. Given this list, execution of `insertSorted(headNode,42)` should result in the list being modified as follows:



For this problem, complete the definition of function `insertSorted` below so it inserts a number into the list in the proper place, so that the resulting list is sorted. If the argument to `insertSorted` is a number that is already in the list, the function should leave the list unchanged, i.e., attempts to insert a duplicate number are ignored. Your code should **not** change the contents of existing nodes. If you insert a new value in the list, you should allocate a new node and splice it into the correct place.

```
struct LNode {       // linked list node
  int data;          // data value
  LNode *next;       // link to next node, or NULL if last node
};

// Insert val in correct place in sorted list beginning at head

// recursive solution

void insertSorted(LNode *head; int val) {

   // precondition: *head is either the header node, or
   //               head->data < val

   if (head->next == NULL || head->next->data > val)
   {
      // insert new node following *head
      LNode *p = new LNode;
      p->data = val;
      p->next = head->next;
      head->next = p;
   }

   else if (head->next->data < val)
      // insert in tail of list
      insertSorted(head->next, val);

   // we get here if head->next->data == val,
   // so return without doing anything
}
```

```
// Insert val in correct place in sorted list beginning at head

// iterative solution

void insertSorted(LNode *head; int val) {

  while(head->next != NULL && head->next->data < val)
    head = head->next;

  if(head->next != NULL && head->next->data == val)
    return;

  LNode* newnode = new LNode;
  newnode->data = val;
  newnode->next = head->next;
  head->next = node;
}
```

Question 4. (6 points) This question involves the use of types `Stack` and `Queue`. Here are the class definitions for stacks and queues of integers.

```
class Stack {            // integer stack:
   Stack();              // construct empty stack
   void push(int k);     // push k on the top of the stack
   int  pop();           // delete item on top of the stack & return
                         //   a copy of it
   int  top();           // return a copy of item on top of the stack
   bool isEmpty();       // true if stack is empty, otherwise false
};

class Queue {            // integer queue:
   Queue();              // construct empty queue
   void insert(int k);   // insert k at the rear of the queue
   int  remove();        // delete item at front of the queue & return
                         //   a copy of it
   int  front();         // return a copy of item at front of the queue
   bool isEmpty();       // true if queue is empty, otherwise false
};
```

Write a program that reads a sequence of integers from `cin` and writes `yes` on `cout` if that sequence of integers is a palindrome and writes `no` if it is not. Recall that a palindrome is a sequence that is the same when read from left to right or from right to left. So the sequence 44 12 64 32 32 64 12 44 is a palindrome, while the sequence 44 12 64 32 32 65 12 44 is not.

Here's the catch: You may ***only use Stacks or Queues*** as defined above, in addition to simple scalar variables (int, bool, etc.). You ***may not*** use arrays, linked lists, trees, or other data structures.

```
#include <iostream>
using namespace std;

// There are several ways to write this - here we use a simple main
// program.  Idea: insert each input value into both a stack and a
// queue.  When all the input is read, extract the data and compare.
// If they match (LIFO and FIFO), it's a palindrome

int main( ) {
   int val;              // current input value
   Stack s;            // stack and queue containing input
   Queue q;

   // copy input to s and q
   while (cin >> val) {
      s.push(val);
      q.insert(val);
   }

   // compare stack and queue.  Both have same # items
   while (!s.isEmpty()) {
      if (s.pop() != q.remove()) {
         cout << "no";
         return 0;
   }
   cout << "yes";
}
```