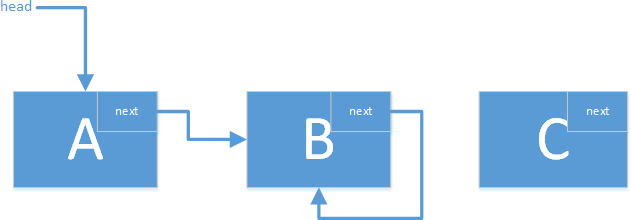
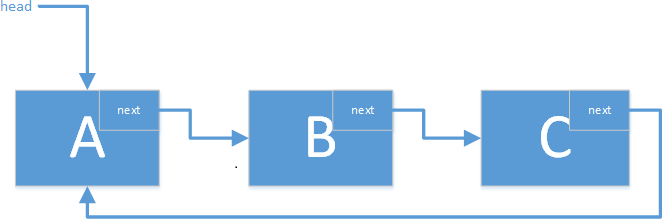
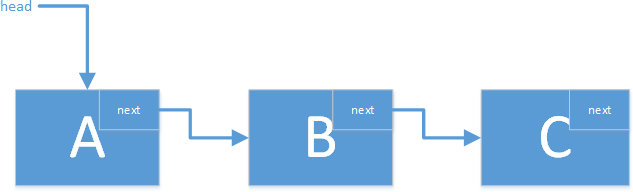
**Module 3: Cairn**

**Question 1 1 pts**

[Acuña] Assume that head is the head node of a singly linked list containing the nodes A, B, and C. What would the result of executing the following code be? Indicate the resulting list using box and arrow notation.

head.getNext().setNext(head.getNext());

1. None of these.
2. A null pointer exception will be thrown.
3. 
4. 
5. 

***Answer***

3.

**Question 2 1 pts**

Suppose that you are asked to analyze the Big-Oh of the following method pop(). What pieces of additional information which need to give a perfectly accurate answer? Select all that apply.

public ItemType pop() {

if(isEmpty())

throw new NoSuchElementException();

ItemType element = top.getElement();

top = top.getNext();

n--;

return element;

}

1. The Big-Oh of the constructor of the NoSuchElementException class.
2. The number of times pop() will be called.
3. The type that will be used in place of ItemType.
4. The Big-Oh of isEmpty().

***Answer***

2,4

**Question 3 1 pts**

Assuming there is a growth function f(n), what does the value of f(0) represent in terms of an algorithm?

***Answer***

f(0) helps us understand the baseline or "fixed" cost of an algorithm when the input size is minimal or zero.

**Question 4 0.5 pts**

What is the Big-Oh order of the following growth expression?

***Answer***

**Question 5 0.5 pts**

Is it *correct* to say that the expression in the previous question is ? Explain.

***Answer***

No; because:

* nlog(n) is **polynomial growth** with a logarithmic factor.
* 2n2^n is **exponential growth**, which increases much faster than any polynomial or logarithmic function.

**Question 6 0.5 pts**

What is the Big-Oh order of the following code fragment? The size of the problem is expressed as n.

for (int i = 1; i <= n; i++)

  for (int j = 1; j <= n; j += 2)

   System.out.println("Nested loops!");  //f(n) counts these

***Answer***

**Question 7 0.5 pts**

What is the Big-Oh order of the following code fragment? The size of the problem is expressed as n.

for (int i = 0; i < (int)Math.pow(2, n); i++)

System.out.println("what could could go wrong?"); //f(n) counts these

***Answer***

**Question 8 0.5 pts**

What is the tilde approximation of the following growth function?

1. Does not exist.

***Answer***

**Question 9 0.5 pts**

You are in the process of choosing between two 3rd party libraries that implement some algorithm and have found two solutions, A and B, that are advertised as O(n) and ~n, respectively. Which of these solutions would you prefer in terms of performance? Explain. (Hint: think about which one gives a more "stable" result.)

***Answer***

In terms of **performance**, you would prefer **Solution B (∼n)**, as it provides more precise and predictable performance characteristics. The tilde notation suggests that the algorithm's behavior is well-analyzed and predictable, which often translates to better and more stable real-world performance.

**Module 5: Cairn**

**Question 1 1 pts**

Consider the following problem, and categorize according to the different axis of problem complexity: create an algorithm to determine which song from a collection to recommend to a user, based on what they have listened to in the past.

It is         [ Select ]      ["open-ended", "close-ended"]  and         [ Select ]      ["ill-defined", "well-defined"]  .

***Answer***

Open-ended and ill-defined.

**Question 2 1 pts**

Understanding the data format and ordering requirements for an algorithm that needs to (as a sub-problem) sort data would take place during         [ Select ]      ["Analysis", "Design", "Justification"]  , while analyzing the sorting algorithm's ability to meet those requirements would happen during         [ Select ]      ["Analysis", "Justification", "Design"]  .

***Answer***

Design, analysis.

**Question 3 1.5 pts**

Consider the following array: 21, 16, 3, 7, 23, 12. Show a trace of execution for**selection sort.** The trace begins with the provided initial state of the array, followed by the array's state after each **swap** is made.

*Enter each subsequent state as a comma separated list (as shown by the initial state). Do not include any spaces. If you follow these directions exactly and are marked off by the auto-grader on the second submission, reach out to the instructional staff.*

initial: 21,16,3,7,23,12

i  =  0: 3,16,21,7,23,12

i  =  1:

i  =  2:

i  =  3:

i  =  4:

i  =  5:

***Answer***

i = 1: 3,7,21,16,23,12

i = 2: 3,7,12,16,23,21

i = 3: 3,7,12,16,23,21

i = 4: 3,7,12,16,21,23

i = 5: 3,7,12,16,21,23

**Question 4 1.5 pts**

Consider the following array: 21, 16, 3, 7, 23, 12. Show a trace of execution for **insertion sort**. The trace begins with the provided initial state of the array, followed by the array's state after each **pass** is made.

*Enter each subsequent state as a comma separated list (as shown by the initial state). Do not include any spaces in your answer. If you follow these directions exactly and are marked off by the auto-grader on the second submission, reach out to the instructional staff.*

initial: 21,16,3,7,23,12

i  =  1: 16,21,3,7,23,12

i  =  2:

i  =  3:

i  =  4:

i  =  5:

***Answer***

i = 2: 3,16,21,7,23,12

i = 3: 3,7,16,21,23,12

i = 4: 3,7,16,21,23,12

i = 5: 3,7,12,16,21,23

**Question 5 1 pts**

Selection sort assumes there is a region of         [ Select ]      ["unsorted elements at the front", "sorted elements at the front", "sorted elements at the back", "unsorted elements at the back"]  of a collection, picks the first unsorted element and places it         [ Select ]      ["in the middle of", "at the beginning", "at the end"]  of the sorted region, until the entire list is sorted.

***Answer***

sorted elements at the front, at the end.

**Question 6 0.5 pts**

What is the Big-Oh order of the following code fragment? The fragment is parametrized on the variable N. Assume that you are measuring the number of assignments to min.

int N = a.length;

for (int i = 0; i < N; i++) {

int min = i; //assignment to min

for (int j = i+1; j < N; j++)

if (less(a[j], a[min]))

min = j; //assignment to min

exch(a, i, min);

}

Group of answer choices

1. Does not exist, or cannot be determined with information given.

***Answer***

**Question 7 0.5 pts**

What is the most efficient sorting algorithm to use for the data set 1, 3, 5, 7, 9 assuming you want to sort the numbers in ascending order.

Group of answer choices

1. Pogosort
2. Shell Sort
3. Selection Sort
4. Insertion Sort

***Answer***

Insertion Sort

**Module 6: Cairn**

**Question 1 1 pts**

Consider the following array: 3, 8, 23, 18, 15, 16. Show a trace of execution for **insertion sort**. The trace begins with the provided initial state of the array, followed by the array's state after each **pass** is made.

*Enter each subsequent state as a comma separated list (as shown by the initial state). Do not include any spaces in your answer. If you follow these directions exactly and are marked off by the auto-grader on the second submission, reach out to the instructional staff.*

initial: 3,8,23,18,15,16

i  =  1: 3,8,23,18,15,16

i  =  2: 3,8,23,18,15,16

i  =  3: 3,8,18,23,15,16

i  =  4: 3,8,15,18,23,16

i  =  5: 3,8,15,16,18,23

**Question 2 0.5 pts**

Match the following scenarios with which sorting algorithm is potentially most appropriate. (Find the best matching, with using each algorithm exactly once.)

1. Datasets that are already almost sorted.

           [ Choose ]             Shellsort             Selection Sort             Mergesort             Insertion Sort

1. Datasets being analyzed on system with a limited number of writes.

           [ Choose ]             Shellsort             Selection Sort             Mergesort             Insertion Sort

1. Large datasets where memory is not a limitation.

           [ Choose ]             Shellsort             Selection Sort             Mergesort             Insertion Sort

1. Datasets where elements are very far away from where they need to be.

           [ Choose ]             Shellsort             Selection Sort             Mergesort             Insertion Sort

***Answer***

1. Insertion Sort
2. Selection Sort
3. Mergesort
4. Shellsort

**Question 3 3 pts**

Consider the following array: 2, 13, 16, 3, 7, 23, 12, 25. Show a trace of execution for top-down mergesort using the method shown in lecture (where both sides are updated at once). Illustrate the contents of the array as it is broken down, and then merged into an ordered state.

*Enter each subsequent state as a comma separated list (as shown by the initial state). Do not include any spaces or brackets in your answer. If you follow these directions exactly and are marked off by the auto-grader, reach out to the instructional staff.*

1 (initial): 2,13,16,3,7,23,12,25 (given as example)

2 (down): 2,13,16,3,7,23,12,25 (given as example; nothing changed!)

3 (down): 2,13,16,3,7,23,12,25 (nothing changed!)

4 (mid): 2,13,16,3,7,23,12,25 (nothing changed!)

5 (up): 2,13,3,16,7,23,12,25

6 (up): 2,3,13,16,7,12,23,25

7 (up): 2,3,7,12,13,16,23,25

**Question 4 0.5 pts**

What is the Tilde approximation order of the following code fragment? Assume that you are measuring the number of assignments to a[], and that n represents the number of indices between lo and hi, inclusive.

public static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi) {

int i = lo, j = mid+1;

for(int k = lo; k <= hi; k++)

aux[k]=a[k];

for (int k = lo; k <= hi; k++) {

if (i > mid) a[k] = aux[j++];

else if (j > hi) a[k] = aux[i++];

else if (less(aux[j], aux[i])) a[k] = aux[j++];

else a[k] = aux[i++];

}

}

1. ~n^2
2. ~n
3. ~(1/2)n
4. ~(1/2)
5. Does not exist, or cannot be determined with information given.

***Answer:*** 2

**Question 5 0.5 pts**

If you need to sort a large dataset on a system with limited memory would it be a good idea to use mergesort? Explain.

1. Yes - mergesort has a cool name so it's guaranteed to run fast.
2. No - mergesort requires a linear number of recursive calls to be made.
3. Yes - mergesort has proven optimal performance.
4. No - mergesort requires an auxiliary array to do the merging step.

***Answer***

4

**Question 6 0.5 pts**

When we talked about insertion sort, we found that it requires only n comparisons and 0 exchanges when the input is sorted. Does this violate the lower bound proof?

1. Yes - this means that some inputs will become unsorted.
2. No - the lower bound proof only applies to mergesort or other merging sorting algorithms.
3. Yes - no sorting algorithm can ever do better than nlogn comparisons.
4. No - a sorted input is a special case, while the lower bound proof applies to all inputs.

***Answer***

4

**Module 7: Cairn**

**Question 1 1.5 pts**

Consider the following array: 8, 9, 17, 4, 3, 20, 25, 5 Show a trace of execution for top-down mergesort using the method shown in lecture (where both sides are updated at once). Illustrate the contents of the array as it is broken down, and then merged into an ordered state.

*Enter each subsequent state as a comma separated list (as shown by the initial state). Do not include any spaces or brackets in your answer. If you follow these directions exactly and are marked off by the auto-grader, reach out to the instructional staff.*

1 (initial): 8,9,17,4,3,20,25,5 (given as example)

2 (down): 8,9,17,4,3,20,25,5 (given as example; nothing changed!)

3 (down):

4 (mid):

5 (up):

6 (up):

7 (up):

***Answer***

3 (down): 8,9,17,4,3,20,25,5

4 (mid): 8,9,17,4,3,20,25,5

5 (up): 8,9,4,17,3,20,5,25

6 (up): 4,8,9,17,3,5,20,25

7 (up): 3,4,5,8,9,17,20,25

**Question 2 0.5 pts**

How (in general) are priority queues related to other sorting algorithms?

Group of answer choices

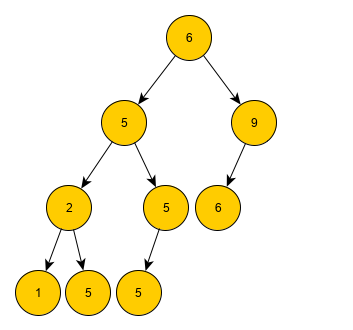
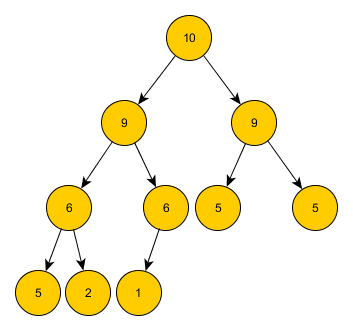
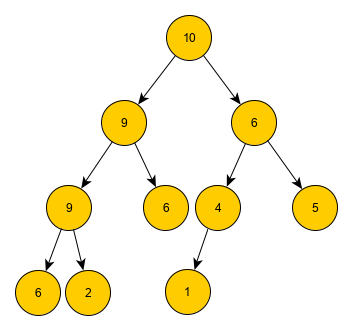
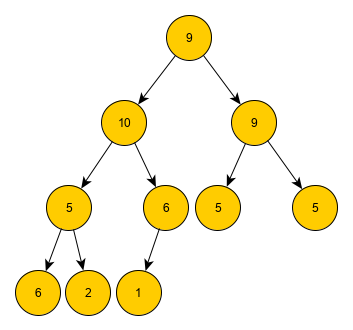
1. Both priority queues and the other four sorting algorithms must obey the lower bound proof.
2. They both involve comparing elements, and PQs may be used to create a straightforward sorting algorithm.
3. Priority queues are implementations of the same ADT that the sorting algorithms use.
4. Priority queues are more efficient implementation of insertion sort.

***Answer***

They both involve comparing elements, and PQs may be used to create a straightforward sorting algorithm.

**Question 3 1.5 pts**

Consider the following values: 5, 9, 1, 5, 6, 2, 5, 6, 9, 10. Select a valid binary heap for this data.

1. 
2. 
3. 
4. 

***Answer***

2

**Question 4 1 pts**

Is your answer to the previous question unique? Explain.

Group of answer choices

1. No - the nodes can also be arranged to fit the BST ordering rule.
2. Yes - these nodes can only be used to draw a specific tree.
3. No - the same nodes can be used to draw multiple valid heaps.
4. Yes - the previous question had only one answer.

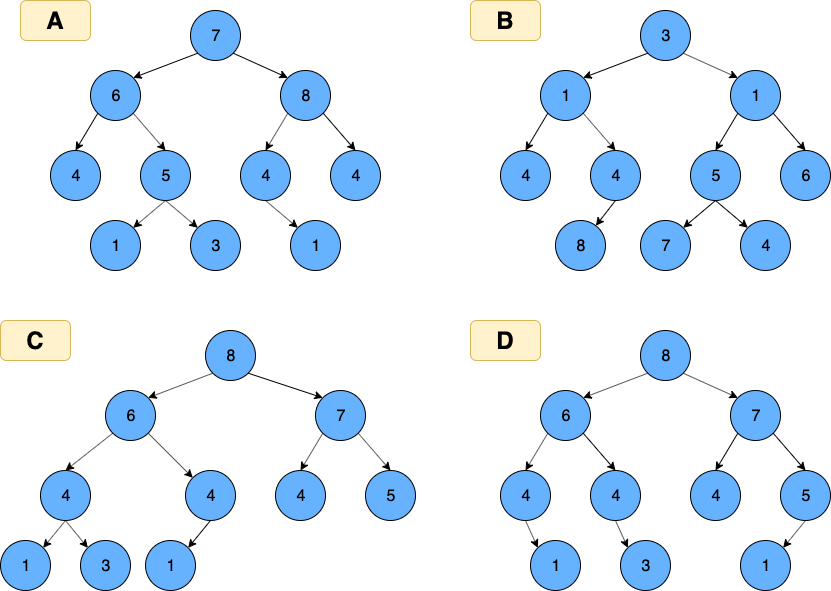
***Answer***

No - the same nodes can be used to draw multiple valid heaps.

**Question 5 1.5 pts**

Consider the following priority queue array: [0, 8, 6, 7, 4, 4, 4, 5, 1, 3, 1]. Which of the diagrams below accurately represents it?

Remember: The first element of the array is never used.



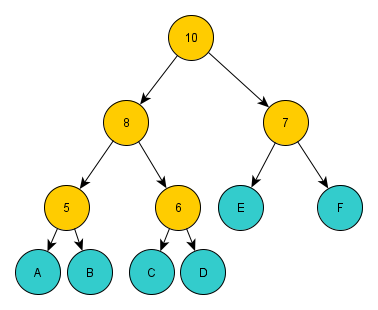
1. A
2. C
3. D
4. B

***Answer***

C

**Question 6 0.5 pts**

Suppose that you are inserting a node with value 4 into the max PQ show below. When it is initially added, which position (see lettered nodes) does it have in the heap/tree?



1. A
2. C
3. F
4. E
5. B
6. D

***Answer***

D

**Question 7 0.5 pts**

What is the Tilde approximation order of the following code fragment? Assume that you are measuring the number of comparisons (*less*) and that n represents the number of nodes in the heap.

private void sink(int k) {

while (2\*k <= N) {

int j = 2\*k;

if (j < N && less(j, j+1)) j++;

if (!less(k, j)) break;

exch(k, j);

k = j;

}

}

1. Does not exist, or cannot be determined with information given.
2. ~(logn)^2
3. ~(1/2)logn
4. ~(1/2)n
5. ~logn

***Answer***

~logn