CS 2420 Practice Exam 100 Points

Fill in the Blank (1 point each)

1. **After many insertions and deletions in a hash table, it is possible that every unused node is marked as "having been deleted". The process used to fix the problem is termed** Rehashing**.**

2. **A** D-arey **is an example of a search tree which is multi-way (allows more than two children).**

3. **The term** Adaptive/non oblivious **refers to a sort that takes advantage of existing order.**

4. **A mergable priority queue which has amortized log n time is a** heap (min/max)**.**

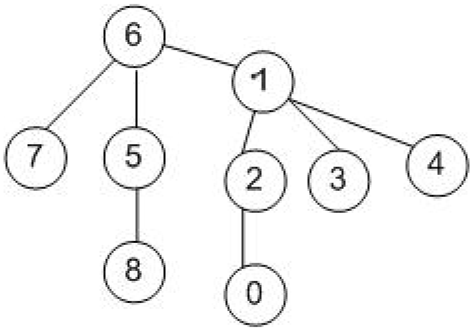
5. I want a search tree in which the worst-case time for a find is log n. A tree that would work is
AVL/Splay(most of the time)/binomial search tree

6. An n log n sort which is stable is merge sort**.**

7. A binomial ? **is a priority queue that is implemented not as a single tree but as a collection of heap-ordered trees.**

Multiple Choice (3 points each)

1. The following disjoint set was generated using union by height (with no path compression). Which of the following series of operations could have generated this tree?



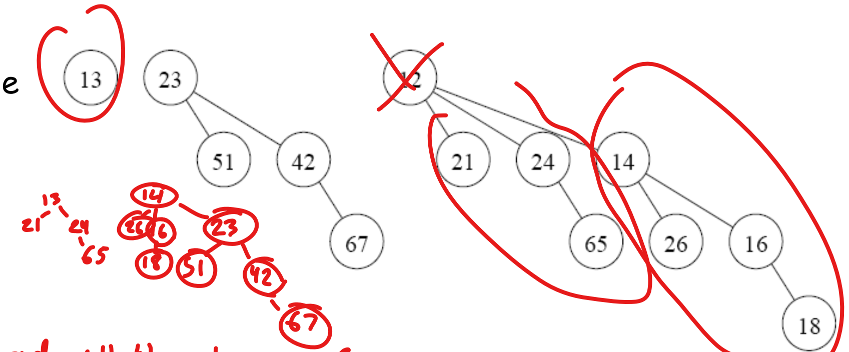
00 00 10 11 01 00 02 12

[✓ a)] union(8,5), union(7,6), union(6,1), union(7,8), union(3,1), union(2,0), union(4,1), union(2,1),
[✓ b)] union(8,5), union(7,6), union(7,8), union(3,1), union(2,0), union(4,1), union(2,1), union(6,1)
[✓ c)] union(3,1), union(8,5), union(7,6), union(7,8), union(2,1), union(2,0), union(4,1), union(6,1)
d) none of the above
e) this tree could not be generated by any set of union by height operations

2. In performing deleteMin on the binomial queue below, what is the result?

2

[✓ a.] the tree rooted at 24 merges with the tree rooted at 13.
b. 13 becomes the parent of 26
c. 21 becomes a single node tree.
d. The tree rooted at 14 becomes split.
e. 14 is swapped into the place of 12.
f. None of the above



We end with three trees of 2^2
Which two do we merge?

3. Consider the following array of ten integers:

5 3 8 9 1 7 0 2 6 4

Suppose we partition this array using quicksort's partition function using 5 for the pivot. Which shows the array after partition finishes:

a. 3 1 0 2 4 5 8 9 7 6
b. 5 3 4 2 1 0 7 9 6 8
[✓ c.] 0 3 4 2 1 5 7 9 6 8
d. 3 1 0 2 4 5 8 9 6 7
e. None of the above

5, 3, 4, 2, 1, 0, 7, 9, 6, 8
0, 3, 4, 2, 1, 5, 7, 9, 6, 8

4. Consider a hash table with quadratic probing and a size of 9. Use the hash function "k%9". Insert the keys: 5, 29, 20, 0, 27 and 19 into your table (in that order). What is the result?

5, 29, 20, 0, 27, 19 K%9

0	27	29	20		5			19
0	1	2	3	4	5	6	7	8

a.

0	27	29	20	5	19			

b.

0	27	29	20	19	5			

[✓ c.]

0	27	29	20		5			19

d.

5	29	20	0			27		19

e. None of the above

5. Two keys that hash onto **different** values originally compete for the same successive locations when resolving collisions. What kinds of collision resolution have this feature?

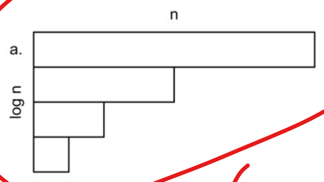
[✓ a.] linear probing
b. double hashing
c. quadratic probing
d. separate chaining
e. a, b, and c
f. none of the above

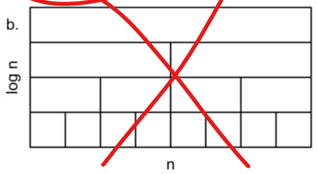
6. You notice slow retrieval from a hash table when the item is not actually in the table. A found item takes an average of 5 probes, but it takes an average of 13.6 probes to determine an item is not in the table. Which factor likely explains the cause of the problem?
   a. The hash function is poor.
   b. The collision resolution method exhibits primary clustering.
   c. The table size is not prime.
   d. Lazy deletion has left most of the keys "deleted."
   [✓ e. All of the above.]

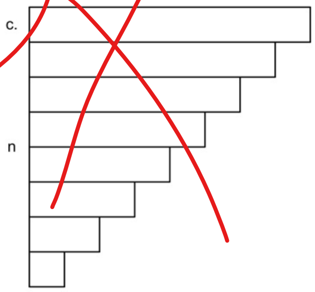
7. You write a hash table implementation that uses separate chaining. What complication of hashing have you avoided?
   ~~a.~~ Bad hash function
   b. Collision Resolution
   ~~c.~~ Rehashing
   d. Lazy deletion
   [✓ e. b and d]
   ~~f.~~ All of the above

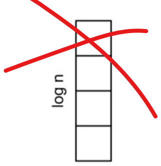
8. For the code below, which picture best represents the complexity?

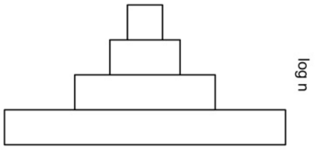
```
void doit (int n)
{  if (n<=1) return;
   for (int  i = 0; i < n; i++ )
        cout << "  ";
  doit(n/2);
} // doit
```

[✓ a.]


b.


c.


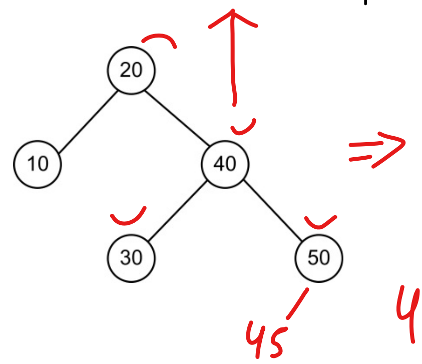
d.


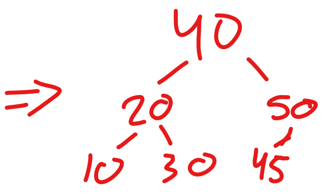
e.


9. The load factor, F, on the hash table indicates rehashing is necessary. The current size of the table is T and the size of the new table is 2T. There are N items currently in the hash table. What is the complexity of rehashing?
   a. O(T)
   b. O(N)
   [✓ c. O(T+N)]
   d. O(F*T)
   e. None of the above.

order O(t) because you have to check if every node needs to be rehashed or not
Then every time we do rehash it is O(1) to rehash so O(t)+O(N) or (T+N)

10. Insert 45 into the AVL tree below and rebalance. What is the preorder traversal of the resulting tree?

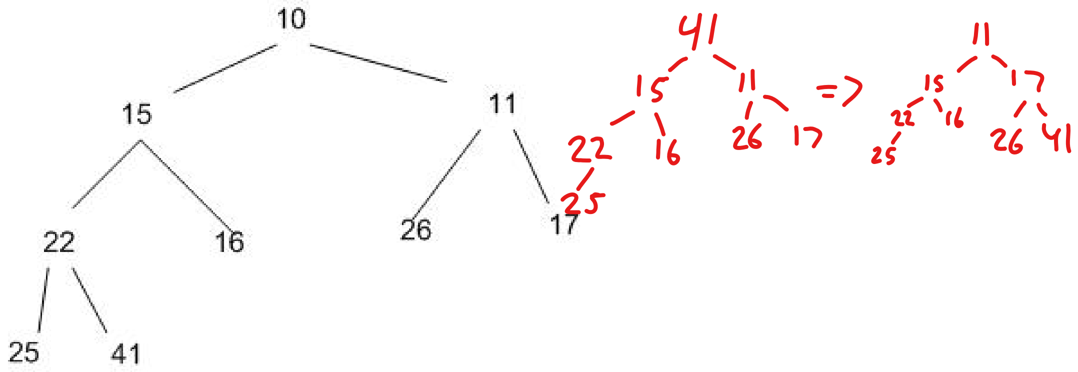




40, 20,10,30,50,45

a. 20 10 45 40 30 50
b. 20 10 50 40 30 45
c. 30 20 10 40 50 45
[✓ d.] 40 20 10 30 50 45
e. 40 20 10 30 45 50
f. none of the above

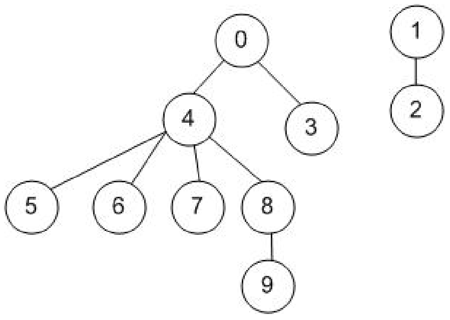
11 . Below is a min heap stored as an array.



Suppose a deleteMin is performed on this heap.
What will (array version of) the heap look like?

[✓ a.] 11 15 17 22 16 26 41 25
~~b.~~ 11 15 ~~16~~ 17 22 25 26 41
c. 11 15 17 22 16 26 25 41
~~d.~~ ~~15~~ 11 17 22 16 26 41 25
e. none of the above

12. The disjoint set shown below was created using what kind of union?



(a) union by height
(b) union by size
(c) neither height or size
(d) either height or size

13. Which of the following nodes need to have its children swapped in order to be a leftist heap?

a. 32 15 10 7

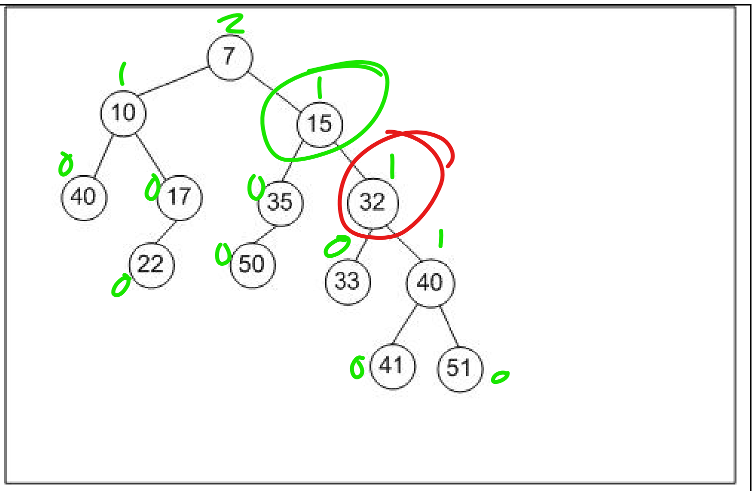
b. 40 32 15 7

c. 32 7

[✓ d. 32 15]

e. 7 10

f. none of the above



1

14.For the timing information below, what is the complexity?

n	$T(n)$
2	6
4	24
8	72
16	192
32	480

a. $O(1)$ b. $O(\log n)$ [✓ c. $O(n)$] d. $O(n\log n)$ e. $O(n^2)$

Short Answer

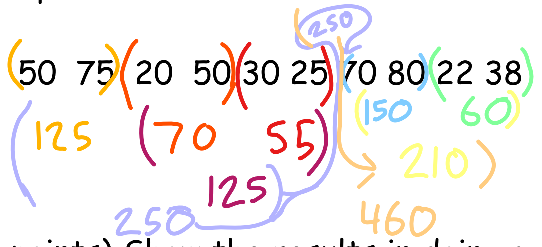
1. (6 points) Timsort is defined as an adaptive merge sort that is a hybrid sort, derived from merge and insertion sort. Explain in what ways it is adaptive and how the hybrid sorts are used together.

- Hybrid [redacted] The mergesort is more time consuming when there are smaller data sets so we use insertion sort until we get to a run that is long enough to effectively use merge sort

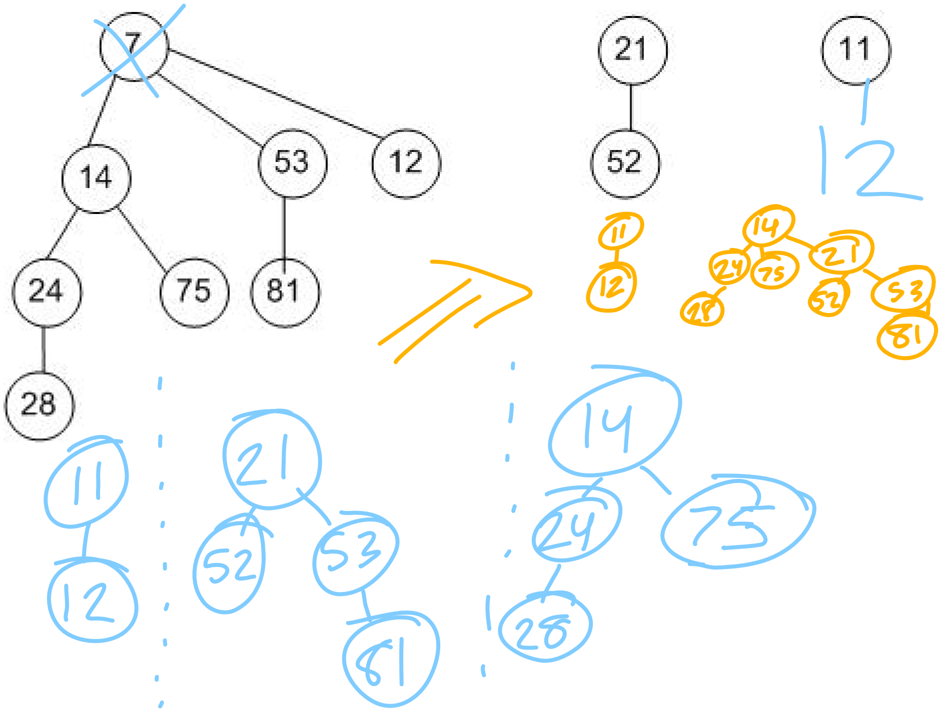
- Adaptive [redacted] Adaptive because we sense when runs of numbers are already in place This is The special characteristic of Tim Sort if the numbers aren't preordered Timsort is only O(nlogn)

2. (6 points) The following represents a series of runs found using Timsort. Timsort merges adjacent runs if the following conditions hold:
   1. |A |<|B|+|C| (merge A B)
   2. |B| <|C| (merge B C)

   Use parentheses to show the order the following runs would be combined:



3. (6 points) Show the results in doing a delete min from the binomial queue below.



4. (8 points) You know multiple ways of implementing a min priority queue. What is the worst case time taken for the following operations?

	AVL Tree	Heap as an array	Leftist heap	Binomial Queue
insertItem	log(n)	log(n)	log(n)	log(n)
findMin	log(n)	O(1)	O(1)	O(1)
RemoveMin	log(n)	log(n)	log(n)	log(n)
Merge queues	O(n)	log(n)	log(n)	O(1)

Does this include merge?

5. (5 points) Quicksort has an expected sorting time of *O*(*n* log *n*) provided that the pivot value is chosen with some care.

- What needs to be true about the choice of pivot value to ensure that quicksort has an expected time of *O*(*n* log *n*)?

As near the median as possible

- Give one effective strategy for picking good pivot values

One way is to look at the beginning, end, and middle values and see which is the median between those 3

6. (16 points) You want to sort an array of Booleans in the shortest time possible. Write the pseudocode to do that. **What is the complexity of your algorithm?**

false		false
false		false
false		false
false		false
true		false
false		false
true	⇨	false
false		false
true		false
false		false
false		true
false		true
true		true
false		true

```
loop until the low search is greater than the high search
    loop looking from low to high for true values (or start from where you left off)
        when one is found hold it in your hand
    loop looking from high to low for false values (or start from where you left off
        when one is found hold it in your other hand
    Interchange the values in your hands
repeat
```

7. (9 points) A min priority queue is stored as an array with the ItemType below.

```
class ItemType
{ public:
   string key;
   int priority;
}
```

The typical operations (insert, deleteMin, makeEmpty) are present. A user needs a new function decreaseKey(string item,int newPriority) which changes the priority of a specific key to a new value. **Describe a good algorithm to do this. What is the complexity of your algorithm?**

Just change the value and swap up or down tree until it finds its spot (assuming all nodes know parents)

log(n)

8. (3 points) A friend says, "A Timsort works well when you have a high degree of sortedness, but is worse than traditional methods when the data is unsorted." Do you agree? Explain.

No, timsort degenerates to nlogn while maintaining a non-oblivious nature as well as stability