Midterm 2011

Summer 2014

on final

Q1 (8 points) What size of a problem can a program handle in given time? The program can use various algorithms with different running time. Fill in the table. (1 ms = 0.001s, 1 μs = 0.000,001s)

algorithm	program running time	1 second	1 hour	1 day	1 week
log n	10 ms for n=1000	10^{300} ✓	$10^{1080000}$ ✓		
n	1 μs for n=1000				
n log n	10 μs for n=1000				
n^2	1 ms for n=1000	$\sqrt{1000 \cdot 1000^2}$ ✓			
2^n	0.5 s for n=10	n=11	n=23 ✓ (approx)		

back / back / back / back

+2 +2 +2

Q2 (12 points) Mark the column for which the statement at the top is true.

f(n)	g(n)	f(n) is Ω(g(n))	f(n) is **O**(g(n))	f(n) is Θ (g(n))
$\sqrt{n^7}$	n^3	✓	✓	
n log n	n $(\log n)^2$		✓	✓
3 n $\log_3$ n	10 n $\log_{10}$ n		✓	✓
2^n	$2^{n+0.001}$		✓	✓
2^n	$(2+0.001)^n$		✓	✓
$(\log n)^2$	n log log n	✓	✓	

Q3 (15 points) Provide the worst-case running time for listed data structures and operations (here p represents a position, e an element, and r the rank in the sequence):

a) Sequence with n elements implemented using an array of Positions where each position stores an element and an array index

addAfter(p,e)	Θ(1)	n
prev(p)	Θ(1)	
remove(p)	Θ(n)	n
indexOf(p)	Θ(1)	
remove(r)	Θ(n)	

b) Sequence with n elements implemented using a doubly-linked list of Positions where each position stores an element

addAfter(p,e)	Θ(n)	
prev(p)	Θ(1)	
remove(p)	Θ(n)	
indexOf(p)	Θ(1)	n
remove(r)	Θ(n)	

86400000 ms/day
log 1000 = 10 ms
log x = 86400000
10 log x = 86400000 $\log_{100}$
log x = 8640000 $\log_{100}$
x = (1000)

Q4 (18 points) What is the time complexity of the algorithms used in following programs?

```
public static int f1(int n) {
      int x = 0;
      for(int i = 0; i < n*n; i += 2)
            x++;
      return x;
}
```

n^2

f1 is Θ(....n.................)

```
public static int f2(int n) {
      int x = 0;
      for(int i = 1; i < n*n; i *= 2)
            x++;
      return x;
}
```

$\log n^2$ or log n after simplifying

- double index everytime

or log n

f2 is Θ(....n.................)

```
public static int f3(int n) {
      int x = 0;
      for(int i = 0; i < n*n*n; i++)
            for(int j = 1; j < n; j *= 2)
            x++;
      return x;
}
```

n^3 log n

$n^3 \log n$

f3 is Θ(....n^2.................)

```
public static int f4(int n) {
      int x = 0;
      for(int i = 0; i < n*n*n; i++)
            for(int j = 1; j < i; j += 2)
            x++;
      return x;
}
```

n^3 divide by 2 jump 2 n^3 run n^3 times

n^6

f4 is Θ(....n^4.................)

```
public static int f5(int n) {
      int x = 0;
      for(int i = 0; i < n*n*n; i++)
            for(int j = 1; j < i; j *= 2)
            x++;
      return x;
}
```

n^3 $\log n^3$

$n^3 \log n^3$

f5 is Θ(....n^4.................)

```
public static int f6(int n) {
      int x = 0;
      for(int i = n; i > 1; i /= 2)   log n
            for(int j = 1; j < n; j *= 2)   n
            x++;
      return x;
}
```

f6 is Θ(....n log n....)

answer at back → answer at back

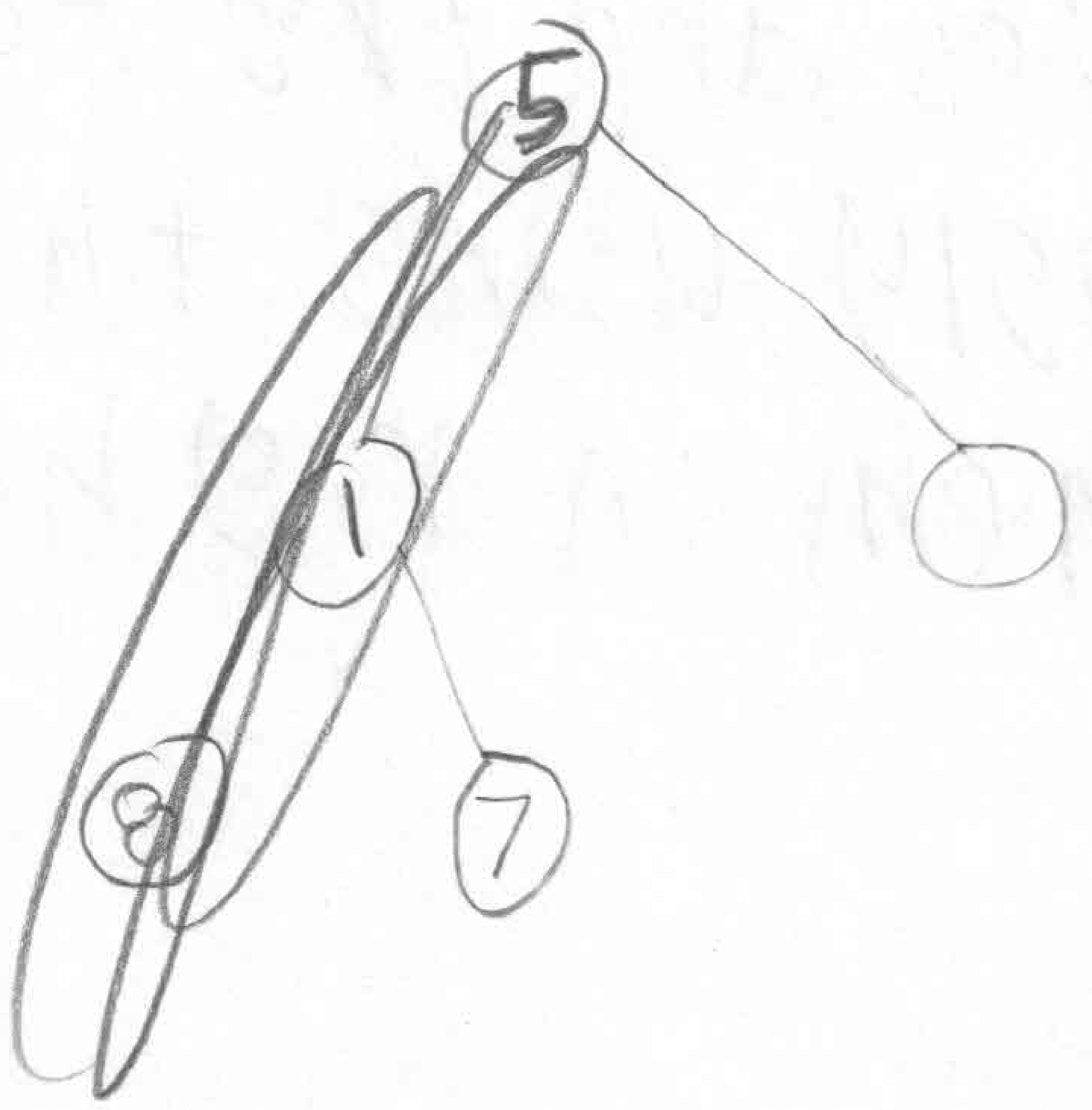
Q5 (16 points) We have a binary tree with elements 1, 2, 3, 4, 5, 6, 7, 8, 9 (not necessarily in this order). Postorder traversal visits the nodes in the order 8, 7, 1, 9, 3, 2, 6, 4, and 5. Inorder traversal visits the nodes in the order 8, 1, 7, 5, 2, 3, 9, 4, and 6.

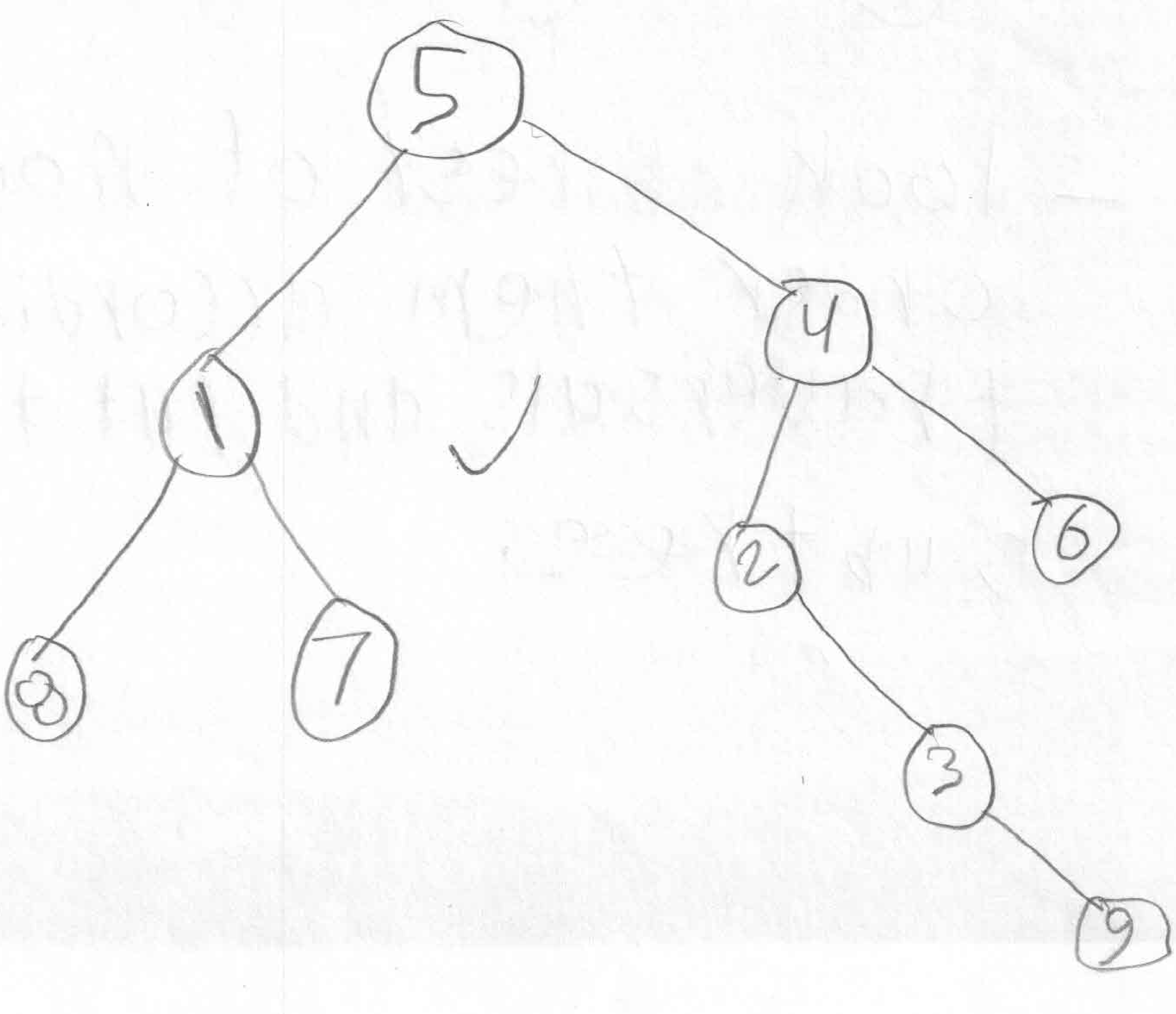
a) Describe a method/strategy to re-construct the tree, preferably as pseudo-code.

1) Pick element from Post order. increment a Postorder index variable (say Post order index) to Pick next element next recursive call.
2) create a new tree node tnode with data picked
3) find the picked element index in inorder (let index be inorder index).
4) call build tree for elements before inorderindex and make buildtree as left subtree of tnode.
5) call build tree for elements after inorderindex and make buildtree as right subtree of tnode. 6) returntnode.

b) Draw the tree (follow the method described above if you have any)

post 8 7 1 9 | 3 2 6 4 5
in 8 1 7 5 | 2 3 9 4 6





Q6 (15 points) Provide short answers:

What are the 3 elements of successful recursion?

- base case (ending condition) ✓
- recursive call
- the ~~call to~~ ~~recur~~ recursive call divides the problem to smaller input & go towards base case.

What are iterators and what are the good reasons to use them?

- iterators are objects that are used to scan a list or a container that contains elements one element at a time. ✓
- reasons to use iterators are to get an element and to traverse a list. ✓

What is a comparator and what are the good reasons to use them?

- the comparator is an object that is external to the class of keys it compares. It has one method compare (a, b)
- reasons to use comparator is when we want to sort a list we can use the comparator to determine which object in list is before or after another object

Can you build a priority queue and adaptable priority queue using an array?

- yes, we can use a heap implemented as an array that uses location aware entries to store its objects. ✓

What are the advantages (give a real example of the benefits) and disadvantages of position-aware entries?

advantages:

- allows for faster insertions and deletions
- allows for faster retrieving of data.

disadvantages:

- Locations must be unique ✓
- More work in terms of implement

Q7 (16 points)

The following array contains a heap with integer keys. Give the final version of the array as it would look after 2 deleteMin() operations:

0	1	2	3	4	5	6	7	8	9	10	11	12
x	3	7	8	40	20	9	10	50	45			

0	1	2	3	4	5	6	7	8	9	10	11	12
x	8	20	9	40	45	50	10					

The following array contains a heap with integer keys. Give the final version of the array as it would look after insert(5) followed by insert(2) operations:

0	1	2	3	4	5	6	7	8	9	10	11	12
x	3	7	8	40	20	9	10	50	45	25		

0	1	2	3	4	5	6	7	8	9	10	11	12
x	2	5	3	40	7	8	10	50	45	25	20	9

Consider a heap implemented as an array A. The heap has 10000 elements stored in A[1] ... A[10000]

True / False There are 5000 leaves

True / False A[4000] > A[3999] don't know in heap inrelation of close nodes.

True / False A[4000] > A[125] heap property

not really unless BST.

True / False In pre-order traversal (as a binary tree) the last visited node contains the largest value

visiting root first so smallest value.

True / False In post-order traversal (as a binary tree) the last visited node contains the smallest value

not really unless BST.

True / False Is it possible to determine if key 99999 is stored in the heap using no more than 14 comparisons?

~~Hashtable.~~

16 operations

BONUS (15 points)

a) Write pseudo code to output a singly-linked list in reverse order when you are NOT allowed to allocate memory dynamically. What is the running time of the algorithm?

b) Write pseudo code to output a singly-linked list in reverse order when you are ALLOWED to allocate memory dynamically. What is the running time of the algorithm?

c) You have an increasingly-sorted circular list (using an array) of n elements that is full. The front and rear pointers have been lost. Explain how you can find the smallest element in better then **O**(n).

a) The running time is $O(n^2)$ because we have to go at end of list & then comming back.

code: Node it = head, Node prev; ~~it~~

1) while (it.next != null) {
prev = it
~~keep go~~ it = it.next }

if (it.next = null)
prev = it

~~while (~~