

CSE 332 : 22Su Midterm

Name:

NetID:	@uw.edu
---------------	---------

Instructions

- The allotted time is **60** minutes. Please do not turn the page until the staff says so.
- This is a closed-book and closed-notes exam. You are not permitted to access electronic devices.
- Read directions carefully, especially for problems that require you to show work or provide an explanation.
- We can only give partial credit for work that you've written down.
- Unless otherwise noted, every time we ask for an O , Ω , or Θ bound, it must be simplified and tight.
- For answers that involve bubbling \bigcirc or \square , make sure to fill in the shape completely: \bullet or \blacksquare .
- If you run out of room on a page, indicate that the answer continues on the back of that page. Try to avoid writing on the very edges of the pages: we scan your exams and edges often get cropped off.
- Make sure you also get a copy of the formula sheet.

Advice

- If you feel like you're stuck on a problem, you may want to skip it and come back at the end if you have time.
- Look at the question titles on the cover page to see if you want to start somewhere other than problem 1.
- Remember to take deep breaths.

Question	Max points
1. Big-O	16
2. Code Analysis	20
3. O , Ω , and Θ , oh, my!	12
4. Write a recurrence	9
5. Solve a recurrence	10
6. AVL	8
7. Heaps	15
8. B-Tree	8
Total	98

1. Big-O [16 points]

For questions asking you about runtime, give a simplified, tight big- \mathcal{O} bound. This means that, for example, $\mathcal{O}(2^{n!})$ or $\mathcal{O}(5n^2 + 7n + 3)$ are unlikely to get points. Unless otherwise specified, all logs are base 2. For short sentence answers, points may be deducted for answers exceeding the sentence limit. You may also leave your answer as an unsimplified formula like $7 \cdot 10^3$ when appropriate.

(a) Best-case runtime finding an element in a sorted array of size n (a) _____

(b) Given a binary search tree of n integers, worst-case runtime to convert the binary search tree into a sorted array. (b) _____

(c) Runtime constructing a binary min heap using Floyd's build heap, given an array of values $n, n - 1, n - 2 \dots 1$, in descending order. (c) _____

(d) Given a B-tree of height 3, with $M=5$ and $L=4$ what is the **MINIMUM** number of data items in the tree? (d) _____

(e) Given a B-tree of height 3, with $M=5$ and $L=4$, what is the **MAXIMUM** number of data items in the tree? (e) _____

(f) The TAs propose a new data data structure - MinThousandHeap, where it's a min heap, but each node has 1000 children.
Claim: MinThousandHeap is always more efficient than a binary heap, since the height of the heap is $\log_{1000} n$ instead of $\log_2 n$, so the runtime for operations such as insert and deleteMin will run much faster.
Do you agree with this claim, why or why not? Explain in 1 sentence.

(g) Name one benefit of using a B-tree over an AVL tree.

(h) When designing a B-Tree, what is a possible downside of storing both the key AND value in the internal nodes for the find operation?

2. Code Analysis [20 points]

Describe the worst-case running time for the following pseudocode functions in Big- O notation in terms of the variable n . Your answer must be tight and simplified. **You do not have to show work or justify your answers for this problem.**

```
(a) int kilimanjaro(int n, int steps) {
    steps = 0;

    for (i = 0; i < 332; i++) {
        steps += n;
    }
    return steps;
}
```

(a) _____

```
(b) int everest(int n, int steps) {
    if (n < 5) {
        System.out.println("Keep climbing!");
        return 0;
    } else if (n < 100) {
        return everest(n / 3, steps);
    } else {
        for (int i = 0; i < n * n * n; i++) {
            steps++;
        }

        return everest(n - 2, steps);
    }
}
```

(b) _____

```
(c) int matterhorn(int n, int steps) {
    int elevation = n;

    while (elevation > 0) {
        elevation = elevation/2;
    }

    return steps;
}
```

(c) _____

```
(d) MinFourHeap rainierAscent(int n) {
    MinFourHeap mountain = new MinFourHeap();

    if (n < 50) {
        return mountain;
    } else {
        for (int i = 0; i < n; i++) {
            mountain.insert(i);
        }
    }

    return mountain;
}
```

(d) _____

```
(e) MinFourHeap rainierDescent(int n) {
    MinFourHeap mountain = new MinFourHeap();

    if (n < 50) {
        return mountain;
    } else {
        for (int i = n; i >= 0; i--) {
            mountain.insert(i);
        }
    }

    return mountain;
}
```

(e) _____

3. \mathcal{O} , Ω , and Θ , oh my! [12 points]

For each of the following statements, indicate whether it is always true, sometimes true, or never true. You do not need to include an explanation. Assume that the domain and codomain of all functions in this problem are natural numbers(1, 2, 3 ...).

(a) If $f(n)$ is $\mathcal{O}(g(n))$ and $f(n)$ is $\Omega(h(n))$ then $f(n)$ is $\mathcal{O}(g(n) \cdot h(n))$.

Always True

Sometimes True

Never True

(b) If $f(n)$ is $\mathcal{O}(g(n))$ and $g(n)$ is $\Omega(h(n))$, then $f(n)$ is $\Theta(h(n))$.

Always True

Sometimes True

Never True

(c) Let $f(n)$ be the best-case running time of a call to `deleteMin` in a binary min heap with n distinct elements. $f(n)$ is $\Omega(1)$.

Always True

Sometimes True

Never True

4. Write a recurrence [9 points]

Give a base case and a recurrence for the runtime of the following function. Use variables appropriately for constants (e.g. c_1 , c_2 , etc.) in your recurrence (you do not need to attempt to count the exact number of operations). **YOU DO NOT NEED TO SOLVE this recurrence**

```
int pirate(int n) {
    if (n < 3) {
        for (int i = 0; i < 10; i++) {
            print("Aye, Aye");
        }
        return 332;
    }
    for (int i = 1; i < pow(2, n); i *= 2) {
        print("Ahoy, matey!");
    }
    return n * pirate(n / 5) + 10 * pirate(n - 2);
}
```

$$T(n) = \begin{cases} \underline{\hspace{10em}} & \text{for } n < 3 \\ \underline{\hspace{10em}} & \text{for } n \geq 3 \end{cases}$$

Yay!! You do **NOT** need to solve this recurrence...

5. Solve a Recurrence [10 points]

Suppose the running time of an algorithm satisfies the recurrence given below. Find the closed form for $T(N)$. **You may assume N is a large power of 2.** Your answer should *not* be in Big- \mathcal{O} notation. Show the exact constants and bases of logarithms in your answer (e.g. do NOT use c_1, c_2 in your answer).

Your final answer must not have any summation symbols or recursion – you may find the list of summations and logarithm identities on the last page of the exam to be useful.

You must show your work to receive *any* credit.

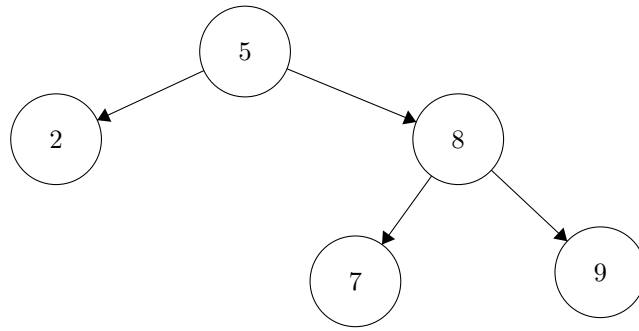
$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T\left(\frac{n}{2}\right) + n & \text{otherwise} \end{cases}$$

6. Adelson-Velskii and Landis [8 points]

(a) (2 points) What is the balance condition for AVL trees? **Answer in one sentence**

(b) (3 points) If we have an AVL Tree with 11 nodes, what is the maximum height of the AVL Tree? Show your work.

(c) (3 points) Suppose we have the following AVL Tree:



Then, we **insert the value 6** into the AVL Tree. What does the AVL Tree look like **AFTER the AVL Tree is balanced**? Make sure to show work for possible partial credit.

7. Heaps [15 points]

- (a) You are busy at your new job writing a binary min-heap implementation. However, your users often get confused on whether low priority numbers mean "urgent" or "not urgent".

So, you decide to write a new operation:

`reverse()`: given a binary min-heap (with root at index 0), convert it to a binary max-heap stored in the same array (with root at index 0)

Your idea to implement this is to just reverse the order of the elements in the array.

ie: If the array backing the heap was [1, 2, 3, 4, 5], it would then become [5, 4, 3, 2, 1]. (Assume the array is exactly the size of the heap).

(5 points) Does this algorithm work? If it does, informally argue why (formal proof is not needed). If it doesn't, provide a counter-example (with explanation of why it is a counter-example)

- (b) (2 points) What are the two properties that define a min-heap. (Not necessarily binary!)

(i)

(ii)

For the next two questions, we are given a BINARY min-heap that contains the elements 1,2,3,...,100.

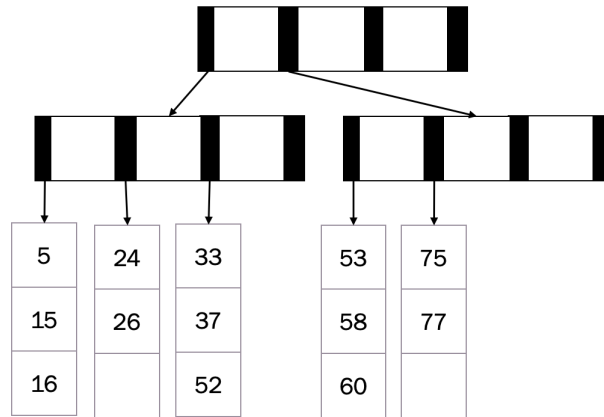
(c) (4 points) Which depth(s) of the heap can the value 4 be found in? Recall the root is at depth 0. For any credit, you must briefly explain your answer.

(d) (4 points) Can the largest value, 100, be found in a non-leaf spot? Why or why not?

8. B-Tree [8 points]

- (a) (2 points) In this ORIGINAL B-Tree shown, **write in the correct values for the internal nodes.**

Original B-Tree:



- (b) (3 points) In the space below, **starting with the ORIGINAL B-Tree**, draw the tree resulting after **inserting the value 19** (including values for internal nodes). Use the method of insertion described in lecture and in the book.

- (c) (3 points) In the space below, **starting with the ORIGINAL B-Tree**, draw the tree resulting after **deleting the value 77** (including values for internal nodes). Use the method of deletion described in lecture and in the book.

Extra piece of paper for scratch work

Reference Sheet

Geometric series identities

$$\sum_{i=0}^k c^i = \frac{c^{k+1} - 1}{c - 1} \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1 - c} \text{ if } |c| < 1$$

Sums of polynomials

$$\sum_{i=0}^n i = \frac{n(n+1)}{2} \quad \sum_{i=0}^n i^2 = \frac{n(n+1)(2n+1)}{6} \quad \sum_{i=0}^n i^3 = \frac{n^2(n+1)^2}{4}$$

Log identities

$$b^{\log_b(a)} = a \quad \log_b(x^y) = y \cdot \log_b(x) \quad a^{\log_b(c)} = c^{\log_b(a)} \quad \log_b(a) = \frac{\log_d(a)}{\log_d(b)}$$

Exponent properties

$$(a^m)^n = a^{m \cdot n} = (a^n)^m$$