CS230 Data Structures **Handout # 24**
Prof. Lyn Turbak **Friday, April 30**
Wellesley College

Problem Set 7
Due: Friday, May 7

Overview: In this problem set, you will get experience with recurrence equations, running times, and asymptotic notations; with sorting; with heaps; and with 2-3 trees. All of the problems on this assignment are pencil-and-paper problems; this problem set requires no programming.

Working Together: You may work with a partner on this assignment. As usual, you are expected to work on all of the problems *together*.

Submission: For each problem, your hardcopy submission should include answers to the pencil and paper problems. There are no softcopy submissions for any problem. Remember to include a signed cover sheet (found at the end of this problem set description) at the beginning of your hardcopy submission.

Problem 1 [10]: Asymptotic Notation

Recall the following from class:

- $\Theta(g)$ is the set of all functions that have the same asymptotic running time as g.
- $O(g)$ is the set of all functions that have an asymptotic running time at most that of g.
- $\Omega(g)$ is the set of all functions that have an asymptotic running time at least that of g.

There are two other relevant notations for comparing asymptotic running times:

- $o(g)$ is the set of all functions that have an asymptotic running time strictly smaller than that of g.
- $\omega(g)$ is the set of all functions that have an asymptotic running time strictly larger than that of g.

Here is an informal summary of the notation:

Notation	Pronunciation	Loosely
$f \in \omega(g)$	f is way bigger than g	$f > g$
$f \in \Omega(g)$	f is at least as big as g	$f \geq g$
$f \in \Theta(g)$	f is about the same as g	$f = g$
$f \in O(g)$	f is at most as big as g	$f \leq g$
$f \in o(g)$	f is way smaller than g	$f < g$

w consider the following six functions:

- $a(n) = 2n \cdot \log_2(n) + n + 5$
- $b(n) = 3n^2 + 7n + 4$
- $c(n) = 17$
- $d(n) = 2^n + 1$
- $e(n) = \log_3(n)$
- $f(n) = 3n - 2$

For each of the following five sets, indicate which of the above functions are members of the set:

1. $O(n)$
2. $o(n^2)$
3. $\Omega(\log_2(n))$
4. $\Theta(n2 + 5n + 2)$
5. $\omega(3^n)$

Problem 2 [15]: Running Times of Membership

Below is a table containing brief descriptions of various data structures that hold collections of integers. For each such data structure, think of the best algorithm for determining if an element is a contained in such a data structure. Then indicate the asymptotic worst-case running time (using Θ notation) of the algorithm in the second column of the table. You should phrase your answer in terms of the number n that appears in each description. Unless the description indicates otherwise, you should not make any assumptions about the arrangement of elements in the data structure.

Description	**Running Time**
An unordered array of n elements.	
An array of n elements ordered from smallest to largest.	
An array of n elements ordered from largest to smallest.	
An n x n two-dimensional array of unordered elements.	
An unordered linked list of n elements.	
A linked list of n elements ordered from smallest to largest.	
A linked list of n elements ordered from largest to smallest.	
A binary tree with n elements.	
A binary search tree with n elements.	
A balanced binary search tree with n elements.	
A complete max heap of n elements.	
A complete min heap of n elements.	
A binary tree of height n.	
A balanced binary search tree of height n.	
A linked list of n balanced binary search trees, each with n elements.	

Problem 3 [18]: Running Time Analysis

In Fig. 1 are six class methods, all of which compute the value of 2 raised to the nth power. For each function, give the following:

1. A recurrence equation that approximates the worst-case running time $T_{\texttt{twoi}}(n)$ of the method `twoi` when invoked on the input n. In all cases, you may assume that $T_{\texttt{twoi}}(n) = 0$ for $n < 1$ and that the overhead of the divide and glue operations within the body of `twoi` has cost that is a constant (for convenience, assume that the constant is 1).

2. A solution to the recurrence equation expressed in Θ notation. You do not have to derive the solutions from scratch. You can just look them up in the table we created in lecture.

Problem 4 [15]: Identifying Sorting Algorithms

Recall the following four algorithms for sorting a collection of n elements:

- **selection sort**: Find the minimum of the n elements, which will be the first element of the sorted collection. Then recursively sort the remaining $n - 1$ elements.

- **insertion sort**: Insert the first element of the collection into the sorted collection that results from recursively sorting the remaining $n - 1$ elements of the collection. .

- **merge sort**: Split the n elements into two collections of with $\lceil \frac{n}{2} \rceil$ and $\lfloor \frac{n}{2} \rfloor$ elements, respectively. Merge the sorted collections that result from recursively sorting the subcollections.

- **quick sort**: Call the first element in the collection the *pivot*, and partition the remaining $n - 1$ elements into two collections: the *lesses* (all the elements $\leq$ the pivot) and the *greaters* (all the elements $>$ the pivot). Then the result of sorting the original collection begins with the result of sorting the lesses, followed by the pivot, followed by the result of sorting the greaters.

When working for AsSorted Systems as a summer intern, Bud Lojack (correctly) implemented the above four sorting algorithms for lists. But instead of giving meaningful names to his routines, Bud named them `sort1`, `sort2`, `sort3`, and `sort4`. When writing up a report at the end of the summer, Bud needs to figure out which routine corresponds to which algorithm. Unfortunately, Bud has accidentally deleted the source files for his routines and can't even inspect the code to determine which routine implements which algorithm. The only the thing he can do is test the routines on various inputs. Fig. 2 presents his timing results for running the routines on input lists of varous sizes. (All times are reported in milliseconds.) For each routine, he has tested the routine on both already sorted lists and on randomly ordered lists.

a. [**8**] Based on Bud's data, fill in the following table with the asymptotic notation that best describes the running time of the routine on a particular type of list. You should use one of the following for every entry of the table: $\Theta(1)$, $\Theta(\log(n))$, $\Theta(n)$, $\Theta(n \cdot \log(n))$, $\Theta(n^2)$, and $\Theta(n^3)$. Note: because the numbers are taken from an actual implementation, they may not fit any category exactly; if the category is ambiguous, say so!

Program	**Sorted List**	**Random List**
`sort1`		
`sort2`		
`sort3`		
`sort4`		

```
public static int two1 (int n) {
  if (n == 0) {
    return 1;
  } else {
    return 2 * two1(n - 1);
  }
}

public static int two2 (int n) {
  if (n == 0) {
    return 1;
  } else {
    return two2(n - 1) + two2(n - 1);
  }
}

public static int two3 (int n) {
  if (n == 0) {
    return 1;
  } else {
    return two3(n - 1) + two1(n - 1); // Yes, the second call is to two1, not two3.
  }
}

public static int two4 (int n) {
  // To simplify analysis, assume that the input to two4 is a power of 2.
  if (n == 0) {
    return 1;
  } else if (n % 2 == 0) {
    int x = two4(n / 2);
    return x * x;
  } else {
    return 2 * two4(n - 1);
  }
}

public static int two5 (int n) {
  // To simplify analysis, assume that the input to two5 is a power of 2.
  if (n == 0) {
    return 1;
  } else if (n % 2 == 0) {
    two5(n / 2) * two5(n / 2)
  } else {
    return 2 * two5(n - 1);
  }
}

public static int two6 (int n) {
  if (n == 0) {
    return 1;
  } else {
    return two6(n - 1) + two4(n - 1) // Yes, the second call is to two4, not two6.
  }
}
```

Figure 1: Six different method for raising 2 to the nth power.

b. [**4**] Based on the table from part a, determine for each of Bud's four routines which of the four sorting algorithms it uses. Briefly explain your reasoning.

c. [**3**] Answer the following for the four sorting algorithms that Bud has implemented:

- Which sorting algorithm(s) does much better on sorted lists than on randomly ordered lists. Why?
- Which sorting algorithm(s) does much worse on sorted lists than on randomly ordered lists. Why?
- Which sorting algorithm(s) takes about the same amount of time on both sorted and randomly ordered lists. Why?

Problem 5 [17]: 2-3 Trees

In the following parts, you are asked to draw some 2-3 trees. As a simple check to avoid errors, verify that every tree you draw is a legal 2-3 tree.

a. [**8**] Draw the sequence of *2-3 trees* $T_0, T_1, \ldots, T_{17}$ that results from inserting the following letters into the empty tree T_0:

`T H E Q U I C K B R O W N Y A M S`

You need only show the final tree that results from each insertion; you need not show the individual steps that lead to each intermediate tree.

b. [**9**] Draw the sequence of *2-3 trees* $T_{17}, T_{18}, \ldots T_{34}$

that results from deleting the following letters one-by-one from the initial tree T_{17}:

`T H E Q U I C K B R O W N Y A M S`

You need only show the final tree that results from each deletion; you need not show the individual steps that lead to each intermediate tree. When deleting a value from a non-terminal node, you may replace it with either its in-order predecessor or in-order successor, whichever is easier.

Problem 6 [25]: A Heap o' Heaps

In the following parts, you are asked to draw some complete *min* heaps (not max heaps) and leftist *min* heaps (not max heaps). As a simple check to avoid errors, verify that every binary tree you draw is a legal complete min heap or leftist min heap.

a. [**5**] Draw as trees the sequence of *complete min heaps* $C_0, C_1, \ldots, C_{17}$ that results from inserting the following letters into the empty heap C_0:

`T H E Q U I C K B R O W N Y A M S`

For example, C_1 should be the result of inserting `T` into the empty heap C_0; C_2 should be the result of inserting `H` into C_1; and so on. Assume that letters appearing earlier in the alphabet are "less than" those appearing later. For example, the root node of C_{17} should be `A`. You need only show the final tree that results from each insertion; you need not show the individual "bubble up" steps that lead to the final tree.

b. [**2**] One advantage of complete min heaps is that they can be represented as arrays. Show how the complete min heap C_{17} would be represented as an array.

```
Time for sort1 to sort sorted list with 400 elements:        4
Time for sort1 to sort sorted list with 800 elements:        8
Time for sort1 to sort sorted list with 1600 elements:      16
Time for sort1 to sort sorted list with 3200 elements:      26

Time for sort1 to sort random list with 400 elements:      255
Time for sort1 to sort random list with 800 elements:      958
Time for sort1 to sort random list with 1600 elements:    4059
Time for sort1 to sort random list with 3200 elements:   16585

Time for sort2 to sort sorted list with 400 elements:       92
Time for sort2 to sort sorted list with 800 elements:      339
Time for sort2 to sort sorted list with 1600 elements:    1356
Time for sort2 to sort sorted list with 3200 elements:    5321

Time for sort2 to sort random list with 400 elements:       13
Time for sort2 to sort random list with 800 elements:       37
Time for sort2 to sort random list with 1600 elements:      71
Time for sort2 to sort random list with 3200 elements:     143

Time for sort3 to sort sorted list with 400 elements:      229
Time for sort3 to sort sorted list with 800 elements:      907
Time for sort3 to sort sorted list with 1600 elements:    3311
Time for sort3 to sort sorted list with 3200 elements:   13634

Time for sort3 to sort random list with 400 elements:      205
Time for sort3 to sort random list with 800 elements:      890
Time for sort3 to sort random list with 1600 elements:    3318
Time for sort3 to sort random list with 3200 elements:   13689

Time for sort4 to sort sorted list with 400 elements:       23
Time for sort4 to sort sorted list with 800 elements:       37
Time for sort4 to sort sorted list with 1600 elements:      87
Time for sort4 to sort sorted list with 3200 elements:     178

Time for sort4 to sort random list with 400 elements:       24
Time for sort4 to sort random list with 800 elements:       51
Time for sort4 to sort random list with 1600 elements:     102
Time for sort4 to sort random list with 3200 elements:     213
```

Figure 2: Timing results for Bud's sorting tests.

c. [**4**] Draw as trees the sequence of *complete min heaps* $C_{18}, C_{19}, C_{20}, C_{21}$ that result from dequeuing the *four* smallest items of C_{17}. You need only show the final tree that results from each deletion; you need not show the individual "bubble down" steps that lead to the final tree.

d. [**4**] Draw as trees the sequence of *leftist min heaps* $L1_0, L1_1, \ldots, L1_8$ that results from inserting the following letters into the empty heap $L1_0$:

T H E Q U I C K

You need not show the intermediate results of individual merge steps.

e. [**4**] Draw as trees the sequence of *leftist min heaps* $L2_0, L2_1, \ldots, L2_9$ that results from inserting the following letters into the empty heap $L2_0$:

B R O W N Y A M S

You need not show the intermediate results of individual merge steps.

f. [**2**] Draw as a tree the *leftist min heap* L_0 that results from merging the two heaps $L1_8$ and $L2_9$.

g. [**4**] Draw as trees the sequence of *leftist min heaps* L_1, L_2, L_3, L_4 that results from dequeuing the *four* smallest items of L_0. You need only show the final tree that results from each deletion; you need not show the individual merge steps that lead to the final tree.

Problem Set Header Page
Please make this the first page of your hardcopy submission.

CS230 Problem Set 7
Due Midnight, Friday, May 7

Name:

Date & Time Submitted:

Collaborators (*anyone you worked with on the problem set*):

By signing below, I attest that I have followed the collaboration policy as specified in the Course Information handout.
Signature:

In the **Time** *column, please estimate the time you spend on the parts of this problem set. Please try to be as accurate as possible; this information will help me design future problem sets. I will fill out the* **Score** *column when grading your problem set.*

Part	**Time**	**Score**
General Reading		
Problem 1 [10]		
Problem 2 [15]		
Problem 3 [18]		
Problem 4 [15]		
Problem 5 [17]		
Problem 6 [25]		
Total		