

CS231 JEOPARDY: THE HOME VERSION
The game that turns CS231 into CS23fun!

Asymptotics

[1] Of the recurrence equations below, indicate *both* of the following:

1. Which gives rise to the best asymptotic running time?
2. Which gives rise to the worst asymptotic running time?
 - a. $T(n) = 1 + T(n - a), a > 0$
 - b. $T(n) = 1 + T(an), 0 < a < 1$
 - c. $T(n) = n + T(n - a), a > 0$
 - d. $T(n) = n + T(an), 0 < a < 1$

[2] In a connected, undirected graph $G = (V, E)$, list *all* of the following that *must* hold:

- a. $|E| = o(|V|)$
- b. $|E| = O(|V|)$
- c. $|E| = \Theta(|V|)$
- d. $|E| = \Omega(|V|)$
- e. $|E| = \omega(|V|)$

[3] List *all* of the following sets that are non-empty:

- a. $o(n) \cap O(n)$
- b. $o(n) \cap \Theta(n)$
- c. $o(n) \cap \Omega(n)$
- d. $o(n) \cap \omega(n)$
- e. $O(n) - (o(n) \cup \Theta(n))$

[4] List *all* of the following sets that are subsets of $O(n^2)$:

- a. $o(n)$ f. $o(n^2)$ k. $o(n^3)$
- b. $O(n)$ g. $O(n^2)$ l. $O(n^3)$
- c. $\Theta(n)$ h. $\Theta(n^2)$ m. $\Theta(n^3)$
- d. $\Omega(n)$ i. $\Omega(n^2)$ n. $\Omega(n^3)$
- e. $\omega(n)$ j. $\omega(n^2)$ o. $\omega(n^3)$

[5] Consider the recurrence $T(n) = n + k \cdot T(n/2)$. For *each* of the following sets, give a value of $k \geq 1$ such that $T(n)$ will have a solution in that set.

- a. $\Theta(n)$
- b. $\Theta(n \lg(n))$
- c. $\Theta(n^2)$

Sorting

[1] What is the best worst-case running time for a comparison-based sort of n numbers?

[2] The quicksort algorithms we studied in class had $\Theta(n \cdot \lg(n))$ expected running time but $\Theta(n^2)$ worst-case running time. Describe a single simple modification to these algorithms that makes them run in $\Theta(n \cdot \lg(n))$ worst-case time.

[3] Consider sorting n numbers (not necessarily distinct) in the range $[1..n]$. List *all* of the following algorithms that have $O(n \cdot \lg(n))$ worst-case running times on this problem:

- a. selection sort f. tree sort
- b. insertion sort h. counting sort
- c. merge sort i. radix sort
- d. heap sort j. general bucket sort
- e. quick sort (random pivot)

[4] Given a heap with n elements, what is the best-case running time for an algorithm that removes all the elements from the heap in sorted order? (*Hint*: Remember Ima Fleik.)

[5] Using an $\Theta(n)$ worst-case linear-time black-box subroutine for finding the median of a set of n numbers, describe a simple linear-time algorithm that selects from a set the element with rank k .

Dynamic Sets

[1] List *all* of the following dynamic set implementations in which the **Search** operation has an $O(\lg(n))$ worst-case running time:

- a. unsorted array f. sorted doubly-linked list
- b. sorted array g. binary tree
- c. unsorted singly-linked list h. binary search tree
- d. sorted singly-linked list i. AVL tree
- e. unsorted doubly-linked list k. red-black tree

[2] List *all* of the following dynamic set implementations whose **Predecessor** operation has an $O(1)$ worst-case running time:

- a. unsorted array
- b. sorted array
- c. unsorted singly-linked list
- d. sorted singly-linked list
- e. unsorted doubly-linked list
- f. sorted doubly-linked list
- g. binary tree
- h. binary search tree
- i. AVL tree
- k. red-black tree

[3] What are the (1) worst-case and (2) expected running times of inserting n distinct elements in random order into the following dynamic set structures?

Structure	Worst Case	Expected Case
sorted list		
binary search tree		
red-black tree		

[4] Consider the following augmentations to each node of a red-black tree. List *all* of the augmentations that *cannot* be maintained without changing the $O(\lg(n))$ worst-case running time of **Insert** and **Delete**.

- a. minimum of the subtree rooted at the node
- b. successor of the node
- c. size of the subtree rooted at the node
- d. depth of the node
- e. height of the node
- f. black-height of the node

[5] Give formulae for *both* the (1) smallest and (2) largest possible number of internal (non-leaf) nodes in a red-black tree with black-height k .

Heaps

[1] Answer *both* of the following:

1. Is every complete heap a leftist heap?
2. Is every leftist heap a complete heap?

[2] What is the largest n such that all the distinct integers in $[1..n]$ can be arranged into a single binary tree that is both a BST and complete heap? Assume that larger numbers have higher priority.

[3] For every n , is it possible that all the distinct integers in $[1..n]$ can be arranged into a single binary tree that is both a BST and a leftist heap? Assume that larger numbers have higher priority.

[4] Answer *all* of the following:

1. What is the worst-case running time for inserting n elements one-by-one into an initially empty complete heap?
2. What is the worst-case running time for building a complete heap from n elements all at once?
3. Are the answers for (1) and (2) different for leftist heaps?

[5] Below is a frequency table for the characters appearing in a message. Draw a Huffman coding tree based on the frequency table.

Character	Frequency
a	6
b	7
c	8
d	9
e	10

Graphs

[1] Answer *both* of the following:

- What is the name of a directed graph that has no back edges?
- What is the name of a directed graph that has no back edges, forward edges, or cross edges?

[2] If G is a directed graph, let \overline{G} be the undirected version of G obtained by erasing arrows on all edges. Let $SCC(G)$ be the number of strongly connected components of G and $CC(\overline{G})$ be the number of connected components of \overline{G} . Which *one* of the following best describes the relationship between SCC and CC for an arbitrary graph G ?

1. $SCC(G) < CC(\overline{G})$
2. $SCC(G) \leq CC(\overline{G})$
3. $SCC(G) = CC(\overline{G})$
4. $SCC(G) \geq CC(\overline{G})$
5. $SCC(G) > CC(\overline{G})$

[3] Draw the smallest connected, undirected graph for which depth-first search induces a different tree from breadth-first search.

[4] Arrange 4 vertices in a DAG such that it has exactly four topological sorts.

[5] Arrange the three vertices A, B, and C into a graph that has exactly two tree edges, two back edges, one forward edge, and no cross edges. Assume that the depth-first search classifying the edges explores vertices in alphabetical order.

Potpourri

[1] True or false: a dynamic programming algorithm that uses a d -dimensional table, each of whose dimensions is size n , runs in $O(n^d)$ time. Briefly justify your answer.

[2] Answer *both* of the following for an arbitrary AVL tree T of height h :

1. What is the maximal number of rotations that can be made during insertion into T ?
2. What is the maximal number of rotations that can be made during deletion from T ?

[3] Of the following greedy algorithms, list *all* that are *not* optimal:

- a. 0/1 knapsack algorithm
- b. fractional knapsack algorithm
- c. coin changing with arbitrary denominations
- d. Huffman coding tree algorithm
- e. LZ78 compression algorithm
- f. Dijkstra's single-source shortest path algorithm

[4] The following game costs \$1 to play: given a coin with probability p of getting heads, flip it twice. If you get two heads you get \$2; otherwise you get nothing. What is the smallest p for which you should play this game?

[5] One way to represent a dynamic set is as a record with two fields: (1) A `size` field indicating the number of elements in the set and (2) An `elts` field containing an array with at least `size` slots, where the slots `elts[1..size]` contain the elements of the set. Insertion is straightforward except when no array slots remain. In this case, a new array whose size is double that of the old array is constructed, and the elements of the old array are copied into the initial slots of the new array before continuing with the insertion.

Assume that updating the fields of a record and slots of an array costs \$1 and constructing an array with n elements costs \$ n . What is the minimal amount of money that `Insert` needs to be charged in order that it has an amortized worst-case constant cost?