University of Waterloo
CS240 Spring 2022
A3 - Solutions

Problem 1 [5 marks]

Derive the average-case run time of the algorithm $HalfSort(A)$. Your derivation must be based on the definition of the average-case running time (not on an equivalence of running time to some randomized version of $HalfSort(A)$). You can assume n is divisible by 2 in your analysis. Express running time asymptotically. You can use the fact that $\sum_{n=0}^{\infty} \frac{n}{2^n} = c$ for some constant c.

```
HalfSort(A)
A: an array of size $n \geq 2$ storing distinct numbers
1.  $i \leftarrow 1$
2.  while $i < n$
3.      if $A[i-1] > A[i]$
4.          return false
5.      $i \leftarrow i + 2$
6.  return true
```

Solution:

For every input of length n where $A[0] < A[1]$, there is an input of length n where $A[1] < A[0]$ and the rest of the elements are identical. Therefore, the algorithm terminates in line 4 during the first iteration, for exactly half of the inputs of length n. Similarly, the algorithm will terminate in line 4 for of the second iteration for half of the remaining half. More generally, for any input where $[A_i - 1] < A[i]$, there is an input where all the elements except $A[i-1]$ and $A[i]$ are the same and $A[i-1] > A[i]$, and we compare elements $A[i-1]$ and $A[i]$ in iteration $\frac{i}{2}$. Hence the sum of running times for all inputs of length n is

$$T^{avg}(n) = \frac{1}{n!} \sum_{A \in \mathcal{I}_n} T(A) = \frac{1}{n!} \sum_{i=0}^{n/2} ci\frac{n!}{2^i} = c \sum_{i=0}^{n/2} \frac{i}{2^i} \leq c \sum_{i=0}^{\infty} \frac{i}{2^i} \leq c.c' \in O(1).$$

Alternative Solution:

Let Π_n denote the set of order permutations of size n.

Note that run time is proportional to the number of comparisons, let $T(\pi)$ denote the number of comparisons needed with input order permutation $\pi \in \Pi_n$.

Hence

$$T^{avg}(n) = \frac{1}{|\Pi_n|} \cdot \sum_{\pi \in \Pi_n} T(\pi)$$

Now, note that i starts with 1 and increments by 2 in each iteration of the for loop, the possible values for the number of comparisons are between 1 and $\frac{n}{2}$.

We have

$$\begin{aligned} T^{avg}(n) &= \frac{1}{|\Pi_n|} \cdot \sum_{\pi \in \Pi_n} T(\pi) \\ &= \frac{1}{n!} \sum_{k=1}^{\frac{n}{2}} k \cdot (\text{the number of order permutations that needs exactly } i \text{ comparisons}) \end{aligned}$$

Now, let π_k denote the number of order permutations that needs at least $i+1$ comparisons

$$\text{the number of order permutations that needs exactly } i \text{ comparisons} = \pi_{k+1} - \pi_k$$

Now, we know $\pi_1 = n!$, as every permutations needs at least 1 comparisons.

For $k = 2$, note that there are $\binom{n}{2}$ ways to choose $A[0]$ and $A[1]$, and once the values for $A[0]$ and $A[1]$ are chosen, we must have $A[0] < A[1]$ as we needs to pass the first comparison to get into the second iteration. There are $\binom{n}{2}$ ways to arrange $A[0]$ and $A[1]$. Once this is done, there are $(n-2)!$ ways to arrange the rest $n-2$ elements. Hence $\pi_2 = \binom{n}{2}(n-2)!$

In general, we have

$$\begin{aligned} \pi_k &= \binom{n}{2} \cdot \binom{n-2}{2} \cdot \ldots \cdot \binom{n-2\cdot(k-2)}{2} \cdot (n-2(k-1))! \\ &= \frac{n\cdot(n-1)}{2} \cdot \frac{(n-2)\cdot(n-3)}{2} \cdot \ldots \cdot \frac{(n-2k+4)(n-2k+3)}{2} \cdot (n-2k+2)! \\ &= \frac{n!}{2^{k-1}} \end{aligned}$$

Consequently,

$$\begin{aligned}
T^{avg}(n) &= \frac{1}{|\Pi_n|} \cdot \sum_{\pi \in \Pi_n} T(\pi) \\
&= \frac{1}{n!} \sum_{k=1}^{\frac{n}{2}} k \cdot (\text{the number of order permutations that needs exactly } i \text{ comparisons}) \\
&= \frac{1}{n!} \sum_{k=1}^{\frac{n}{2}} k \cdot (\pi_{k+1} - \pi_k) \\
&= \frac{1}{n!} \sum_{k=1}^{\frac{n}{2}} k \cdot (\frac{n!}{2^{k-1}} - \frac{n!}{2^k}) \\
&= \sum_{k=1}^{\frac{n}{2}} \frac{k}{2^k} \\
&< \sum_{k=0}^{\infty} \frac{k}{2^k} = c
\end{aligned}$$

Overall, $T^{avg}(n) \in O(1)$.

Problem 2 [2+2+4=8 marks]

Consider the algorithm below, where *random(k)* returns an integer from the set of $\{0, 1, 2, \dots, k-1\}$ uniformly and at random.

```
ArrayAlg(A)
A: an array storing numbers
    if A.size == 1
        return
    i ← random(A.size)
    for j = 0 to i do
        print(A[j])
    ArrayAlg(A[0, ..., A.size − 2])
```

a) What is the best-case (i.e. the best luck) running time of *ArrayAlg* on array A of size n? Justify.

b) What is the worst-case (i.e. the worst luck) running time of *ArrayAlg* on array A of size n? Justify.

c) Let $T^{exp}(n)$ be the expected running time of $ArrayAlg$ of size n. Show how to derive a recurrence relation for $T^{exp}(n)$ and then solve it. Express $T^{exp}(n)$ using big-O asymptotic notation. Your bound must be asymptotically tight, but you need not prove that it is tight.

Solution:

- The best-case running time is in $\Theta(n)$. This happens if the the random function in line 3 generates 0 every single time. In that case, the *for* loop in line 4 will take constant time. On the other hand, each time the recursive call is happening on an array with one less element, the running time in the best case scenario is $T(n) = c + T(n-1), T(1) = 1$. Hence the best-case running time is in $\Theta(n)$.

- The worst-case running time is in $\Theta(n^2)$. This happens if the the random function in line 3 generates $A.size - 1$ every time. In that case, the running time of the algorithm is $T(n) = n + T(n-1), T(1) = 1$. Hence the worst-case running time is in $\Theta(n^2)$.

- We know that if size of A is 1, then $T(A, R) = 1$. Else $T(A[0, \cdots, n-1], R) = T(A[0, \cdots, n-1], < i, R' >) = 1 + 1 + c \cdot i + T(A[0, \cdots, n-2], R')$. For an integer $n > 1$ and constant $c \geq 1$:

$$
\begin{aligned}
\sum_R \Pr(R) T(A[0, \cdots, n-1], R) &= \sum_{<i,R'>} \Pr(< i, R' >) T(A[0, \cdots, n-1], < i, R' >) \\
&\leq \sum_{i=0}^{n-1} \frac{1}{n} \sum_{R'} \Pr(R')(2 + ci + T(A[0, \cdots, n-2], R')) \\
&\leq \frac{1}{n} \sum_{i=0}^{n-1} \left(2 + ci + \sum_{R'} \Pr(R') T(A[0, \cdots, n-2], R') \right) \\
&\leq 2 + \frac{1}{n}(c(\frac{n(n-1)}{2})) + \frac{1}{n} \sum_{i=0}^{n-1} \sum_{R'} \Pr(R') T(A[0, \cdots, n-2], R') \\
&\leq 2 + \frac{c(n-1)}{2} + \frac{1}{n} \sum_{i=0}^{n-1} \max_{A \in \mathcal{I}_{n-1}} \sum_{R'} \Pr(R') T(A[0, \cdots, n-2], R') \\
&\leq 2 + \frac{c(n-1)}{2} + \frac{1}{n} \sum_{i=0}^{n-1} T^{exp}(n-1) \leq 2 + \frac{c(n-1)}{2} + T^{exp}(n-1).
\end{aligned}
\tag{1}
$$

This holds for all arbitrary values of A, including the one that maximizes $\sum_R \Pr(R) T(A[0, \cdots, n-1], R)$, which is $T^{exp}(n)$. Hence we get

$$T^{exp}(n) \leq 2 + \frac{c(n-1)}{2} + T^{exp}(n-1)$$

We claim that $T^{exp}(n) \leq cn^2$. To prove this claim, we use induction. The base case is $T(1) = 1 \leq c$, which is correct. Let's $T^{exp}(n-1) \leq c(n-1)^2$, then we have:

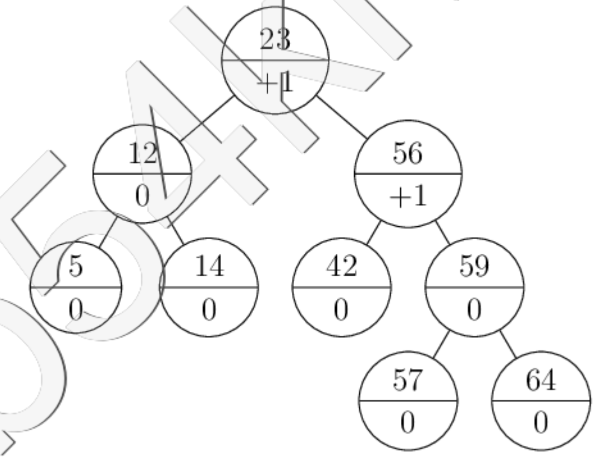
$$\begin{aligned} T^{exp}(n) &\leq 2 + \frac{c(n-1)}{2} + T^{exp}(n-1) \leq 2 + \frac{c(n-1)}{2} + c(n-1)^2 \\ &\leq 2 + \frac{c(n-1)}{2} + cn^2 - 2cn + c \leq cn^2 - \frac{3cn}{2} + \frac{c}{2} \\ &\leq cn^2 \text{ since } c \geq 1 \text{ and } n > 1 \\ &\Rightarrow T^{exp}(n) \leq cn^2 \end{aligned} \tag{2}$$

Problem 3 [0 + 3 + 3 + 3 = 9 marks]

Work with an AVL tree:

a) **Practice** (not worth any marks): Starting with an empty AVL tree, insert the following keys in order: 42 5 12 14 23 56 57 59 64. You should obtain the AVL tree given in the next part.

b) Given the following AVL tree: Note: this tree shows balance factors instead of height.

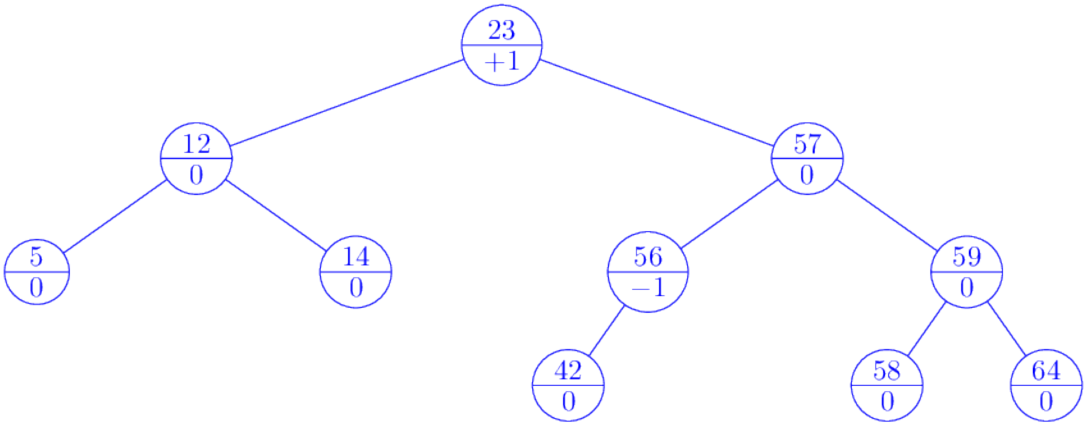


Insert the following keys in order: 58⋆, 16, 15, 22⋆. Show the resulting AVL trees with balance factors (not height) for each node after the elements marked with star (⋆) are inserted.

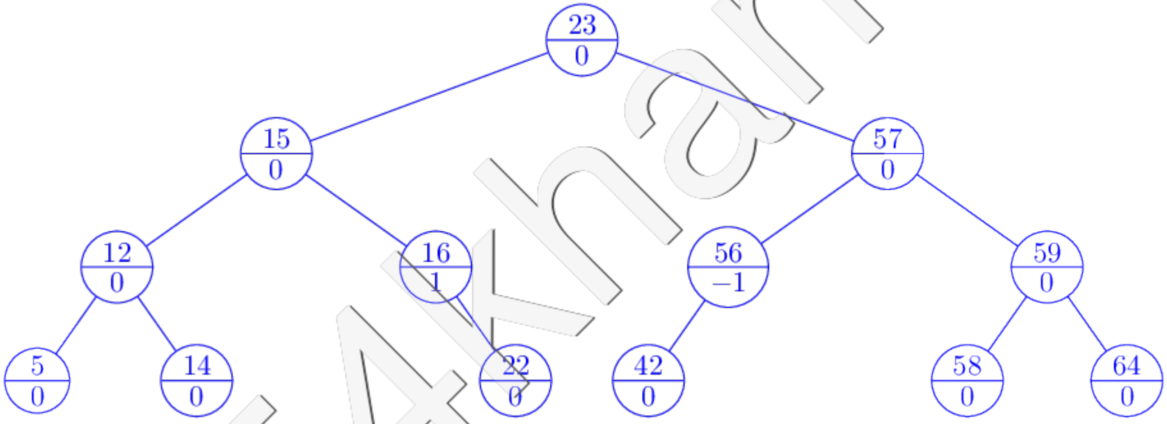
Note: you should only show 2 trees.

Solution:

- After 58



- After 22



c) Consider all the nodes in the tree we get at the end of part (b). Suppose we start with an empty tree. What is an ordering for inserting the nodes into the tree that does not cause any rotations? **Write** your example, then **propose** a general idea of creating such orderings and **justify** your answer.

Solution: 23, 15, 57, 12, 16, 56, 59, 5, 14, 22, 42, 58, 64

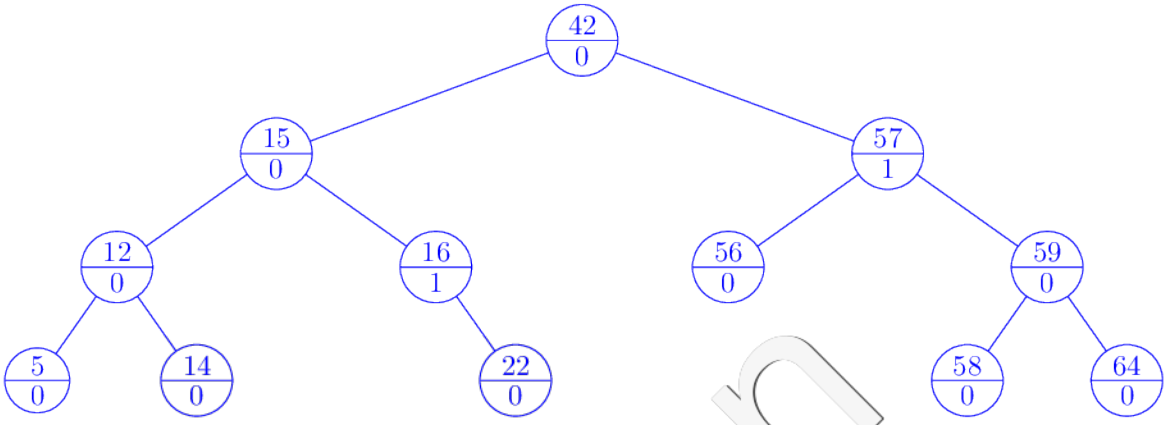
to find such orderings we need to enter nodes in the order of levels from root to leaves. Since insert add nodes as leaves, if a node appears before its children, there will be no need for a rotation. To achieve this, we can recursively apply quick select on the partitions to find the median, or we can alternatively sort the numbers, and at iteration i (starting from 1), insert the numbers at indices $\frac{n(2j-1)}{2^i}, j \in 1, \cdots, 2^i - 1$.

d) Delete the following keys in order: 23⋆, 12, 14, 5⋆.
Show the resulting AVL trees with **balance factors** (not height) for each node after the elements marked with star (⋆) are deleted. If you have a choice of which element to move up, pick the inorder successor.

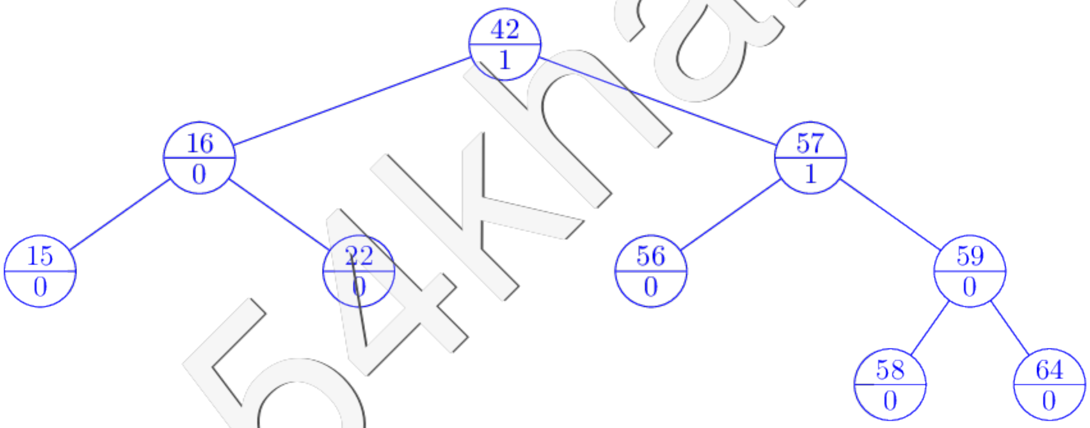
Note: you should only show 2 trees.

Solution (delete from the new tree):

- After 23

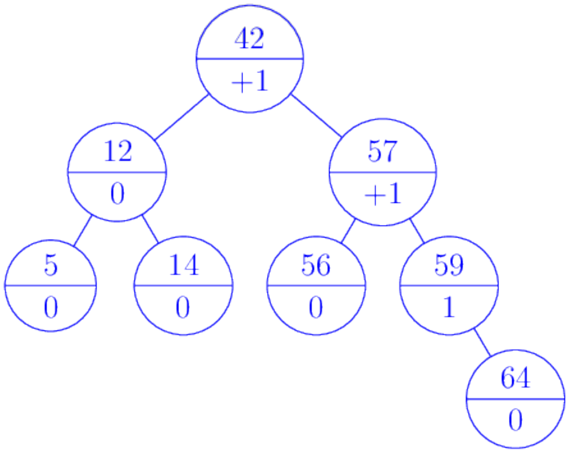


- After 5

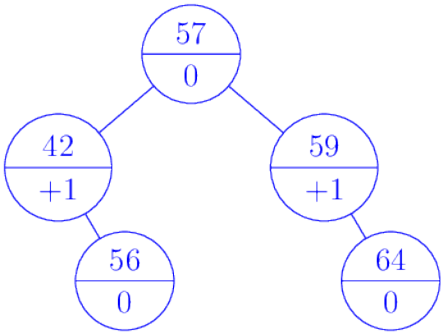


Solution (delete from the original tree, using a double-left rotation):

- After 23

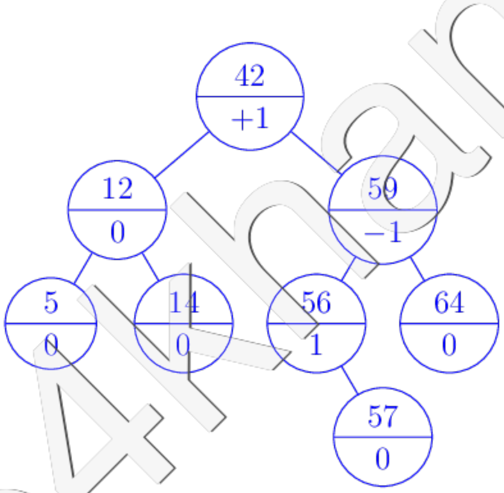


- After 5

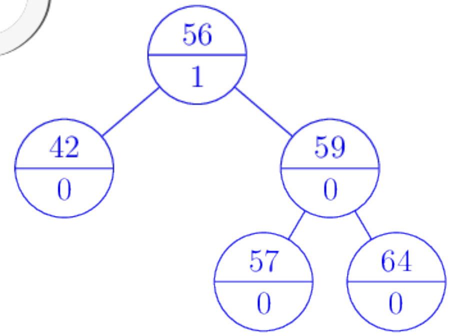


Solution (delete from the original tree, using a single-right rotation):

- After 23



- After 5



Note: With deletions, the restructure that uses fewer rotations (single instead of double) is always the recommended option.

Problem 4 [10 marks]

Given two *AVL trees T1* and *T2*, where the largest key in *T1* is less than the smallest key in *T2*, *Join(T1,T2)* returns an *AVL tree* containing the union of the elements in *T1* and *T2*. Give an algorithm (in pseudocode) for *Join* that runs in time $O(\log n)$, where n is the size of the resulting *AVL tree*. Justify the correctness and efficiency of your algorithm.
Assume the height of *T1* is greater than or equal to the height of *T2*; the other case is symmetric.

Solution: We solve this in two steps, first, for the scenario where the heights of *T1* and *T2* differ by at most 1.

Since $h_2 \leq h_1$, let z be the largest key in $T1$. Using AVL Delete, remove z from $T1$ and call the resulting tree TC.

Create a new tree with root z, left subtree TC and right subtree $T2$.

Ordering: all nodes in the left subtree are smaller than root z, all nodes in the right subtree are larger than z and both TC and $T2$ are AVL trees.

Structure: After removing z, trees differ in height by at most one. Resulting tree is the AVL since subtrees of z differ by at most one and subtrees are AVL.

Runtime: Finding z and removing it is $O(h)$ and creating the new tree is $O(1)$ time so the algorithm takes $O(h)$ time.

If the difference between the height of the trees is more than one, then, we

1. Search $T1$ for a rightmost subtree of height h_2, call this subtree TC and the parent of the root of the subtree p.
2. Remove TC from $T1$ and merge TC with $T2$, as in part a), call the resulting tree TD.
3. Replace the right subtree of p in $T1$ with TD.
4. Rebalance from p if needed.

Details that need to be addressed.

- $T1$ may not contain a rightmost subtree TC of height h_2. If this does not exist, i.e. p may be the root of a subtree of height $h_2 + 1$ with a left subtree of height h_2 and right subtree of height $h_2 - 1$. Choose the rightmost subtree TC of height $h_2 - 1$.
- Tree TD may have height h_2 or $h_2 + 1$ after removing a node from $T2$ and making it the new root. When replacing TD as the left subtree of p in $T1$, this may result in a new height for p and my require rebalancing. All elements of TC are smaller than elements of $T2$ so when merged, resulting tree TD is also AVL.

Ordering: All elements of TD are greater than p so replacing the right subtree with TD maintains the ordering property.

Height of the new left subtree of p may be 1 greater than in the initial tree so p may require rebalancing but will then be AVL. Each operation is $O(h)$ so the runtime is $O(h)$.

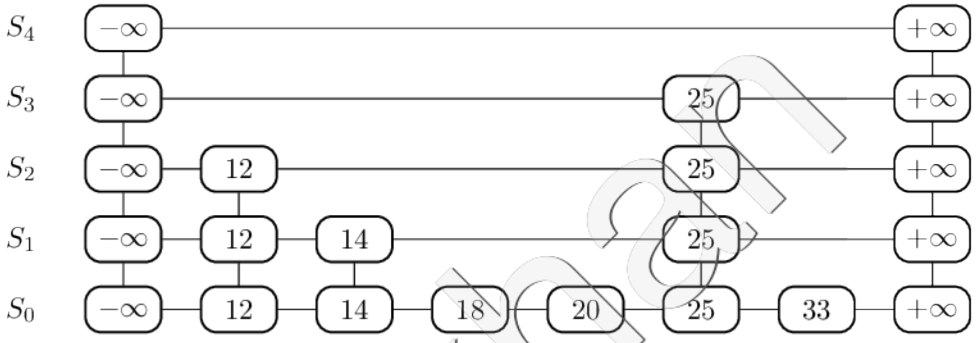
Problem 5 [0 + 3 + 3 + 3 + 3 = 12 marks]

a) Practice (not worth any marks): Starting with an empty skip list, insert the following keys in order: 18 12 33 25 14 20 using the following coin flip sequence:

THHTTHHHTHTTHT

You should obtain the skip list given in the next part.

b) Given the following skip list:

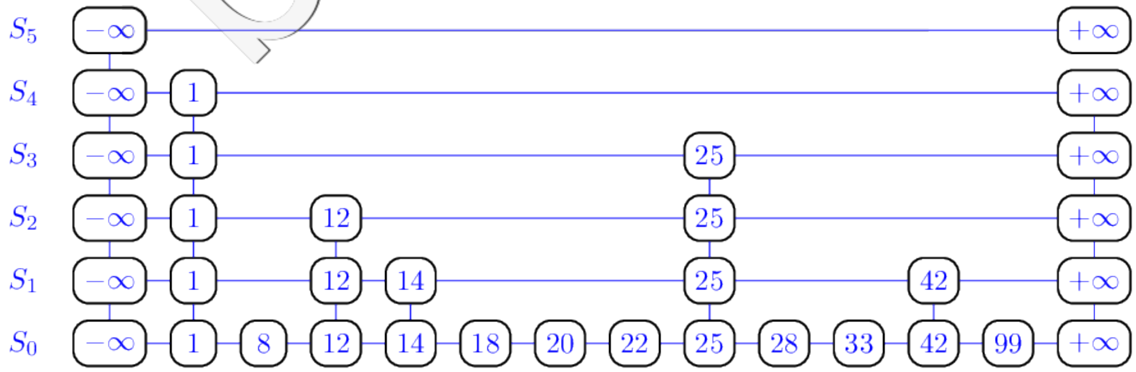


Show the skip list that is created by inserting the keys: 28, 42, 8, 1, 22, 99 into the given skip list. You must use the following coin flip sequence:

$$THTTHHHHTTTTHTTHT$$

Note that each coin flip in the sequence will only be used once, in order and there may be some unused coin flips.

Solution:



After inserting all keys, determine the exact number of comparisons (between 2 keys) required to search for the keys inserted in this part (that is, indicate what the search cost would be for 28, and then the search cost for 42, and so on).

For example, a search for 18 in the given skip list above (from Part (a)), requires 6 comparisons.

Key	28	42	8	1	22	99
Comparisons	8	10	7	6	11	9

For the remaining parts, assume that the probability of adding a level to a tower is p $(0 < p < 1)$, as opposed to $\frac{1}{2}$.

c) Explain why the probability that a tower in the skip list with height at least i is p^i.

Solution: For a tower to have height at least i, the first i flips must be heads. The probability of each flip is p and they are independent so p^i.

d) Show that the expected number of extra nodes (i.e., the total number of nodes in skip list not including the nodes in S_0) is $O(n)$. Therefore, the space requirements for this skip list are linear in the number of keys being stored.

Solution: The expected tower height of a key is $p/(1-p)$ (not including the node at level S_0. So the expected number of keys on level i is np^i for $i \geq 1$. Sum over all levels (i goes to infinity), to obtain $np/(1-p)$ which is $\Theta(n)$. Also, each level has 2 sentinel nodes but the expect the height of the skiplist to be $\Theta(\log n)$ so this does not effect the previous result.

e) Show that the expected height of a skip list is at most $\log_{1/p}(n) + 1/(1-p)$.
Remember to also consider the left and right sentinels.
To make the math a bit easier, you may assume n is a power of $1/p$.

Solution: Let v_i be the indicator variable with $V_i = 0$ if level i is empty and 1 otherwise. Since $v_i \leq 1$ then $E(V_i) \leq 1$. V_i is always less than or equal to the number of nodes at level i. Then, the expected value of V_i is less than or equal to the expected number of nodes at level i which is np^i by part b); thus, $E(V_i) \leq np^i$.

The height of the skip list is equal to $(\sum_{i\geq 0} V_i) - 1$ or $\sum_{i \geq 1} V_i$.

Then the expected height is:

$$
\begin{aligned}
E\left((\sum_{i \geq 0} V_i) - 1 \right) &= \left(\sum_{i \geq 0} E(V_i) \right) - 1 \text{ by linearity of expectation} \\
&= \left(\sum_{i=0}^{\log_{1/p} n - 1} E(V_i) \right) + \left(\sum_{i \geq \log_{1/p} n} E(V_i) \right) - 1 \\
&\leq \log_{1/p} n + \left(\sum_{i \geq \log_{1/p} n} np^i \right) - 1 \text{ since } E(V_i) \leq 1 \text{ and } E(V_i) \leq np^i \\
&= \log_{1/p} n + np^{\log_{1/p} n} \left(\sum_{i \geq 0} p^o \right) - 1 \\
&= \log_{1/p} n + n \left(p^{\frac{\log_{1/p} n}{\log_p}(1/p)} \right) \left(\frac{1}{1-p} \right) - 1 \text{ by the given formula} \\
&= \log_{1/p} n + n \left(p^{\log_p n} \right)^{-1} \left(\frac{1}{1-p} \right) - 1 \\
&= \log_{1/p} n + n(n)^{-1} \left(\frac{1}{1-p} \right) - 1 \\
&= \log_{1/p} n + \left(\frac{1}{1-p} \right) - 1
\end{aligned}
\tag{3}
$$