Table of Contents

Question 1.

Let T be the random variable for the final value of t.

Let R_i be the number of times **`random(1, 2^c)`** is called when $c = i$.

Notice ΔT_i (change in T during the i^{th} iteration) is equal to R_i.

R_i is a geometric distribution, since it runs until it gets a success.

The probability of a random number from 1 to 2^c being equal to 1 is $\dfrac{1}{2^i}$

So $E[R_i] = 2^i$ by the expected value of a geometric distribution.

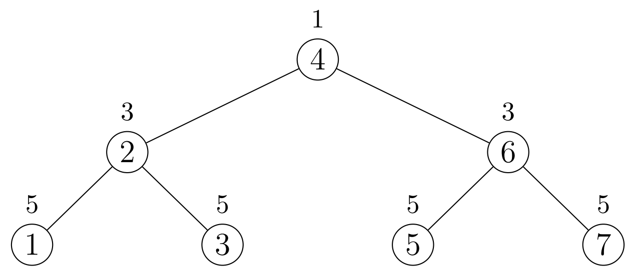
The outer while loop loops until $c > n$, so T is just the sum of all R_i.

$$\begin{aligned} E[T] &= \sum_{i=1}^{n} E[R_i] \\ &= \sum_{i=1}^{n} 2^i \\ &= 2(2^n - 1) \end{aligned}$$

Question 2.

(a)

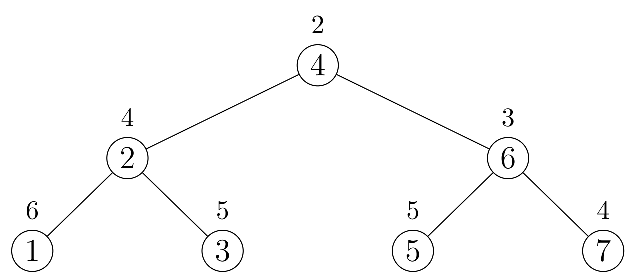
Since there are only 7 nodes, I can manually calculate the number of comparisons for each node.



Since node k is chosen at random, then every node has a $\frac{1}{7}$ chance of being selected. So

$$
\begin{aligned}
E[\text{Comparisons}] &= \frac{1}{7}(1) + \frac{1}{7}(3) + \frac{1}{7}(3) + \frac{1}{7}(5) + \frac{1}{7}(5) + \frac{1}{7}(5) + \frac{1}{7}(5) \\
&= \frac{27}{7}
\end{aligned}
$$

(b)

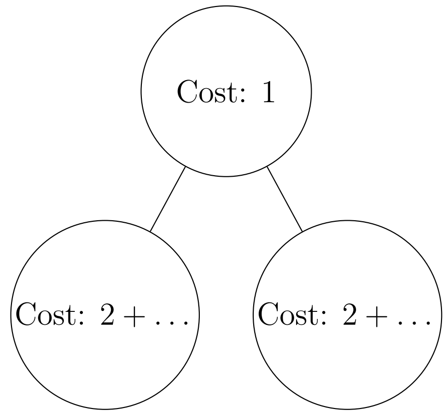


$$
\begin{aligned}
E[\text{Comparisons}] &= \frac{1}{7}(2) + \frac{1}{7}(4) + \frac{1}{7}(3) + \frac{1}{7}(6) + \frac{1}{7}(5) + \frac{1}{7}(5) + \frac{1}{7}(4) \\
&= \frac{29}{7}
\end{aligned}
$$

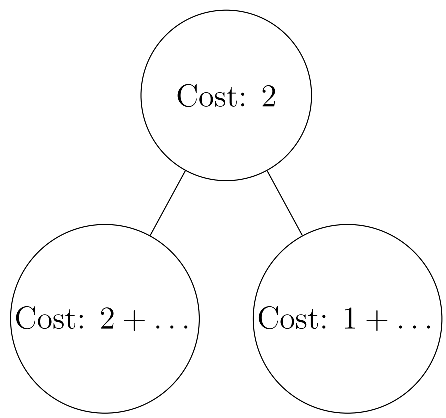
(c)

I think that for bigger trees, **`SEARCH2(r, k)`** uses less comparisons than **`SEARCH1(r, k)`**.

This is because for **`SEARCH1(r, k)`**, it takes 1 comparison if the current node is **`k`**, and **`2 + rchild(r)`** to go left, and **`2 + lchild(r)`** to go right.



For **`SEARCH2(r, k)`**, It takes 2 comparisons if the current node is **`k`**, and **`2 + rchild(r)`** to go left, and **`1 + lchild(r)`** to go right.



This means that the more you go right, the more comparisons you start to save. So for bigger trees, you'll save more than the small trees, where you only go right 3 times.

Question 3.

(a)

Operation 1	Operation 2	Probability
`PREPEND(x, S)`	`PREPEND(x, S)`	p^2
`PREPEND(x, S)`	`ACCESS(S, 1)`	$p \times \frac{1-p}{2}$
`PREPEND(x, S)`	`ACCESS(S, 2)`	$p \times \frac{1-p}{2}$
`ACCESS(S, 1)`	`PREPEND(x, S)`	$p \times (1-p)$
`ACCESS(S, 1)`	`ACCESS(S, 1)`	$(1-p)^2$

(b)

Let $L_i = \begin{cases} 1 & \text{if the } i^{\text{th}} \text{ operation is } \texttt{PREPEND} \\ 0 & \text{if the } i^{\text{th}} \text{ operation is } \texttt{ACCESS} \end{cases}$

The probability of any operation being **PREPEND** is p.
Since L_i is an indicator random variable, then $E[L_i] = p$

Let X = length of S. S is just the sum of all L_i plus one since it starts with one element.

$$\begin{aligned} E[\text{S}] &= 1 + \sum_{i=1}^{k-1} L_i \\ &= 1 + \sum_{i=1}^{k-1} p \\ &= 1 + (k-1)p \end{aligned}$$

(c)

Let X_k = number of steps required to perform the k^{th} operation

Let A_k = number of steps needed to perform `ACCESS` during the k^{th} operation.

so $X_k = \begin{cases} 1 & \text{if the } k^{\text{th}} \text{ operation is } \texttt{PREPEND} \\ A_k & \text{if the } k^{\text{th}} \text{ operation is } \texttt{ACCESS} \end{cases}$

A_k is a uniform distribution over $\{1, \cdots, |S|\}$

The expected value of $|S|$ is $1 + (k-1)p$, so

$$\begin{aligned} E[A_k] &= \frac{1 + 1 + (k-1)p}{2} \\ &= 1 + \frac{(k-1)p}{2} \end{aligned}$$

$$\begin{aligned} E[X_k] &= \begin{cases} 1 & \text{if the } k^{\text{th}} \text{ operation is } \texttt{PREPEND} \\ E[A_k] & \text{if the } k^{\text{th}} \text{ operation is } \texttt{ACCESS} \end{cases} \\ &= \begin{cases} 1 & \text{if the } k^{\text{th}} \text{ operation is } \texttt{PREPEND} \\ 1 + \dfrac{(k-1)p}{2} & \text{if the } k^{\text{th}} \text{ operation is } \texttt{ACCESS} \end{cases} \\ &= 1(p) + \left(1 + \frac{(k-1)p}{2}\right)(1-p) \\ &= p + 1 + \frac{(k-1)p}{2} - p - \frac{(k-1)p^2}{2} \\ &= 1 + \frac{p(k-1)(1-p)}{2} \end{aligned}$$

(d)

Let Y = number of steps needed for n operations

$$\begin{aligned}
E[Y] &= \sum_{i=1}^{n} X_i \\
&= \sum_{i=1}^{n} 1 + \frac{p(i-1)(1-p)}{2} \\
&= \sum_{i=1}^{n} 1 + \sum_{i=1}^{n} \frac{p(i-1)(1-p)}{2} \\
&= n + \frac{p(1-p)}{2} \sum_{i=1}^{n} (i-1) \\
&= n + \frac{p(1-p)}{2} (-n + \sum_{i=1}^{n} i) \\
&= n + \frac{p(1-p)}{2} (-n + \frac{n+n^2}{2}) \\
&= n - \frac{np(1-p)}{2} + \frac{np(1-p)(n+1)}{4} \\
&= \frac{4n}{4} - \frac{2np(1-p)}{4} + \frac{np(1-p)(n+1)}{4} \\
&= \frac{4n - 2np(1-p) + np(1-p)(n+1)}{4} \\
&= \frac{4n - 2np + 2np^2 + (np - np^2)(n+1)}{4} \\
&= \frac{4n - 2np + 2np^2 + n^2p - n^2p^2 + np - np^2}{4} \\
&= \frac{4n - np + np^2 + n^2p - n^2p^2}{4}
\end{aligned}$$

Question 4.

(a)

If S_i is **`INSERT`**

$C_{1,i}$ always takes $\mathcal{O}(1)$ comparisons since the node is inserted at the front of the list.

$C_{2,i}$ takes on average $\dfrac{1}{1-a}$ comparisons, where a is the load factor of the bucket.

If S_i is **`DELETE`**

$C_{1,i}$ takes the same or less than amount of comparisons as $C_{2,i}$. This is because for both hash tables, you must traverse the bucket to delete the node. In this case, there will be $\mathcal{O}(a)$ comparisons for both cases.

$C_{1,i}$ could take less comparisons than $C_{2,i}$ because you could **`DELETE`** a node in the middle of the bucket. This will shorten the bucket for $C_{1,i}$, but not for $C_{2,i}$.

(b)

If you insert 2 keys **`a`** and **`b`** with the same hash, then T_1 will have a linked list **`b -> a`**. However in T_2, it will have **`a, b`** for the buckets. So **`SEARCH(a)`** will take less comparisons for T_2 than T_1.

(c)