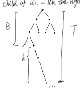
Question 1:

Construction: Fix n. Choose an integer k such that $k=\omega(\log n)$ and $k^2=o(n\log n)$ (eg: $k=(\log n)^2$ works for large n)

1. Build a balanced binary search tree B with $n-k$ nodes. Its height is $O(\log n)$.
2. Select one leaf v of B, which lies at depth $O(\log n)$
3. Attach a chain of length k to v be repeatedly adding a right child: let u_1 be the right child of v, u_2 the right child of u_1, … u_k the right child of u_{k-1}



This gives a BST T with n nodes. The height is height (T) = height $(B)+k$ = height $(B)+\omega(\log n)$

Let $D(T)$ be the sum of depth of all nodes.

Balanced part B: Each of the $(n-k)$ nodes has depth $O(\log n)$. $D_B=O((n-k)\log n)=O(n\log n)$

Chain part: The first chain node has depth $O(\log n)+1$, the second $O(\log n)+2$, …, the kth node $O(\log n)+k$

$D_{chain}=k\cdot\log n+\sum_{i=1}^{k} i=O(k\log n)+O(k^2)$

Total sum of depths: $D(T)=D_B+D_{chain}=O(n\log n)+O(k\log n)+O(k^2)$

$\because k^2=o(n\log n)$ and $k\log n=O(n\log n)$, the last 2 terms are lower-order. $\therefore D(T)=O(n\log n)$

$\therefore$ the average depth is $\frac{D(T)}{n}=O(\log n)$

$\therefore$ the best T satisfies:

average node depth: $O(\log n)$

height: $O(\log n)+k=\omega(\log n)$

This is exactly the type of BST the problem asks for.

Question 2:

(a) $P(E_1|E_3)=\frac{P(E_1\cap E_3)}{P(E_3)}=\frac{1\cdot\binom{4}{2}}{\binom{5}{3}}=\frac{6}{10}=\frac{3}{5}$

(b) $P(E_3|E_2)=\frac{P(E_2\cap E_3)}{P(E_2)}=\frac{\binom{4}{2}}{2^4}=\frac{6}{16}=\frac{3}{8}$

(c) $P(E_2|E_3)=\frac{P(E_2\cap E_3)}{P(E_3)}=\frac{\binom{4}{2}\cdot 1}{\binom{5}{3}}=\frac{3}{5}$

(d) $P(E_3|E_1\cap E_2)=\frac{P(E_3\cap E_1\cap E_2)}{P(E_1\cap E_2)}=\frac{3}{2^3}=\frac{3}{8}$

(e) $P(E_1)=\frac{2^4}{2^5}=\frac{1}{2}$, $P(E_2)=\frac{2^4}{2^5}=\frac{1}{2}$

$P(E_1\cap E_2)=\frac{2^3}{2^5}=\frac{1}{4}$

$P(E_1)\times P(E_2)=P(E_1\cap E_2)$ $\therefore E_1$ and E_2 are independent

(f) $P(E_2)=\frac{1}{2}$, $P(E_3)=\frac{\binom{5}{3}}{2^5}=\frac{10}{32}=\frac{5}{16}$

$P(E_2\cap E_3)=\frac{\binom{4}{2}}{2^5}=\frac{6}{32}=\frac{3}{16}$

$P(E_2)\times P(E_3)=\frac{1}{2}\times\frac{5}{16}=\frac{5}{32}\neq\frac{3}{16}$

$\therefore E_2$ and E_3 are not independent

Question 3:

First, use the elementary binomial identity

$k\binom{n}{k}=n\binom{n-1}{k-1}$ $\left(n\cdot\frac{(n-1)!}{(k-1)!(n-k)!}=k\frac{n!}{k!(n-k)!}\right)$

$\therefore \sum_{k=1}^{n}k\binom{n}{k}^2=\sum_{k=1}^{n}n\binom{n-1}{k-1}\binom{n}{k}=n\sum_{k=1}^{n}\binom{n-1}{k-1}\binom{n}{k}$

For $\sum_{k=1}^{n}\binom{n-1}{k-1}\binom{n}{k}$ Let $j=k-1$.

$1\le k\le n\rightarrow 0\le j\le n-1$

$\therefore \sum_{k=1}^{n}\binom{n-1}{k-1}\binom{n}{k}=\sum_{j=0}^{n-1}\binom{n-1}{j}\binom{n}{j+1}$

$=\sum_{j=0}^{n-1}\binom{n-1}{j}\binom{n}{n-j-1}$ $\left(\because \binom{n}{j+1}=\binom{n}{n-j-1}\right)$

Using Vandermonde's identity

$\binom{a+b}{n}=\sum_i\binom{a}{i}\binom{b}{n-i}$ $a=n-1$, $b=n$

$\therefore \sum_{j=0}^{n-1}\binom{n-1}{j}\binom{n}{n-j-1}=\binom{n-1+n}{n-1}=\binom{2n-1}{n-1}$

$\therefore \sum_{k=1}^{n}k\binom{n}{k}^2=n\binom{2n-1}{n-1}$

Question 4:

Define a subproblem:

Let maxEndingHere[i] = maximum sum of a subarray that ends at index i

Then we can write the recurrence:

maxEndingHere[i] = max(a_i, maxEndingHere[i-1] + a_i)

explanation: either the maximum subarray ending at i consists only of a_i (the previous elements are all negative), or it extends the maximum subarray ending at i-1.

Base case: maxEndingHere[1] = a_1

Here is the pseudocode:

```
def max_subarray(A):
    n=len(A)
    maxEndingHere = A[0]
    maxSoFar = A[0]
    for i in range(1, n):
        maxEndingHere = max(A[i], maxEndingHere+A[i])
        maxSoFar = max(maxSoFar, maxEndingHere)
return maxSoFar
```

The subproblem graph has n vertices and n-1 edges.

vertices: each subproblem maxEndingHere[i] for i=1,2,…,n

edges: maxEndingHere[i] depends on maxEndingHere[i-1]

if there are n vertices, then we have n-1 edges

maxEndingHere[1] → maxEndingHere[2] → … → maxEndingHere[n]

Question 5:

1. problem setup

activity-selection problem:

input: A set of activities $A=\{a_1, a_2, \dots a_n\}$, each with a start time s_i and finish time f_i

goal: Select the maximum number of mutually compatible activities (no overlaps)

classic optimal greedy strategy:

always pick the activity with the earliest finish time that is compatible with previously selected activities

this strategy is provably optimal.

alternative greedy strategy (fails sometimes):

always pick the activity with the shortest duration (f_i-s_i) that is compatible with previously selected activities

2. counterexample

consider the following activities:

activity	start	finish	duration
A_1	1	10	9
A_2	2	3	1
A_3	3	4	1
A_4	4	5	1

Greedy by shortest duration:

1. Pick A_2 2. Pick A_3 3. Pick A_4

Total activities selected: 3

Optimal strategy (earliest finish time)

1. Pick A_1 (duration = 9)

A_2, A_3, A_4 all overlap with A_1 → cannot pick.

Total activities selected: 1

The "shortest-duration-first" strategy only considers local optimality (current shortest activity)

It can block future selections that would allow more activities to be chosen.

The correct greedy strategy is earliest-finish-time-first, which guarantees globally optimal solutions.

Conclusion:

1. Not every greedy strategy works
2. Shortest-duration-first may fail because local optimal choices do not always lead to the global optimum
3. Earliest-finish-time-first is the provably correct greedy strategy for the activity-selection problem

Question 6:

1. problem setup:

We have n software licenses

Constraint: At most 1 license per month

Price growth: $\begin{cases}\text{license } j \text{ currently costs } 100\\ \text{in } t \text{ months, license } j \text{ will cost:}\\ P_j(t)=100\cdot r_j^t,\ r_j>1\end{cases}$

Goal: Determine the purchase order to minimize total spending.

2. key observation

Since each month you buy exactly one license, the total spending is: Total Cost $=\sum_{t=0}^{n-1}100\cdot r_{\text{license bought at month } t}^{t}$

Notice that licenses with higher growth rates r_j increase in price faster over time.

To minimize total cost, we should buy the fastest-growing licenses first.

Greedy strategy: Sort all licenses in descending order of growth rate r_j and buy them in that order, one per month.

3. Greedy Algorithm

```
def minimize_cost(r):
    n=len(r)
    r_sorted = sorted(r, reverse = True)
    total_cost=0
    for month in range(n):
        total_cost += 100 * (r_sorted[month] ** month)
    return total_cost, r_sorted
```

4. proof of optimality

claim: Buying licenses in decreasing order of r_j minimizes total cost.

Proof (exchange argument):

① Suppose we have an optimal schedule that does not buy licenses in descending order.

② Then there must exist 2 consecutive months i and $i+1$ such that: $r_i<r_{i+1}$ but license with r_i is bought first.

③ Swap these 2 licenses. Let $t=0$. Original cost for these 2 months: $C_{orig}=100\cdot r_i^t+100\cdot r_{i+1}^{t+1}$

④ Cost after swapping: $C_{swap}=100\cdot r_{i+1}^t+100\cdot r_i^{t+1}$

Difference: $C_{orig}-C_{swap}=100(r_i^t+r_{i+1}^{t+1}-r_{i+1}^t-r_i^{t+1})$

$=100(r_{i+1}^t(r_{i+1}-1)-r_i^t(r_i-1))$

Since $r_{i+1}>r_i>1$ $\therefore r_{i+1}^t(r_{i+1}-1)>r_i^t(r_i-1)$

$\therefore C_{swap}<C_{orig}$

Therefore, descending order of r_j is optimal.

5. Running time

Sorted n licenses: $O(n\log n)$

Computing total cost (loop over n months): $O(n)$

overall running time: $O(n\log n)$