Quiz 1

Algorithm performs 28 n^2 + 30 n log^2 n + 78 operations. Determine its asymptotic time complexity (among all valid options select the most commonly used notation)

- ○ O(28 n^2 + 30 n log^2 n + 78)
- ◉ O(n^2)
- ○ O(n^2 + n log^2n)
- ○ O(28 n^2)

All constants are insignificant in big O notation, so this expression is equivalent to $n^2 + n\log^2 n$, moreover, since $O(n^2) > O(n\log^2 n)$ we can drop the second term, the final result is $\boldsymbol{O(n^2)}$

What is the time complexity of this algorithm? Please provide the most accurate estimation

```
var count = 0;
for (int i = 1; i < n*n; i *= 3)
    count++;
```

- ○ O(n^2)
- ○ O(n)
- ○ O(1)
- ○ O(n log n)
- ◉ O(log n)
- ○ O(log^2 n)

This loop iterates over values $1, 3, 9, ..., 3^k < n^2$, where $3^{k+1} \geq n^2$. So we make k operations, and the value of k is: $\log(3^k) < \log(n^2) \Rightarrow k < \log(n^2) = 2\log(n) \Rightarrow k = O(\log(n))$

What is the time complexity of this function? Please provide the most accurate estimation

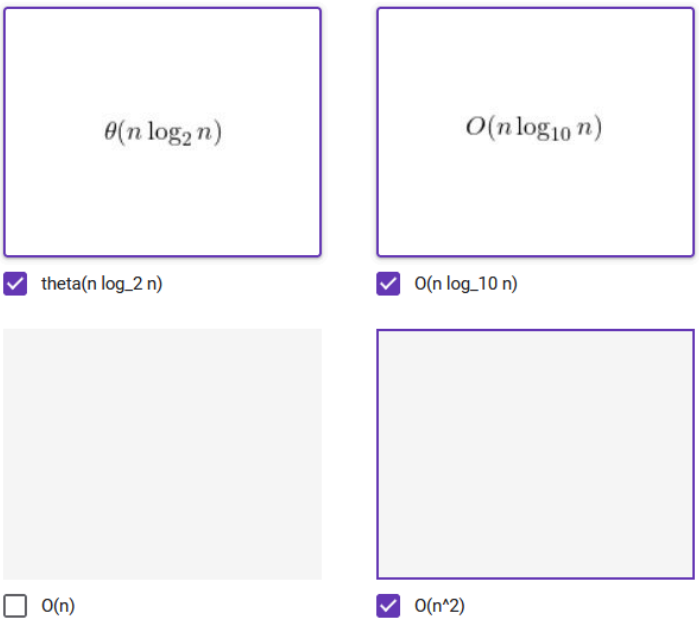
```
foo(arr):
    n = length(arr)

    for i from 0 to n-1:
        swapped = false

        for j from 0 to n-1-i:
            if arr[j] > arr[j+1]:
                swap(arr[j], arr[j+1])
                swapped = true

        if not swapped:
            break

    return arr
```

- (●) O(n^2)
- ( ) O(n log n)
- ( ) O(n)
- ( ) O(n^3)

This is bubble sort implementation, time complexity of which is $\boldsymbol{O(n^2)}$

f(n) = n + n log_10 n
Select all classes to which f(n) belongs

- [ ] $\Omega(n^2)$ — Omega(n^2)
- [x] $\Omega(n)$ — Omega(n)
- [x] $\Omega(n \log_{10} n)$ — Omega(n log_10 n)
- [x] $\theta(n \log_{10} n)$ — theta(n log_10 n)



Intuitively, omega is "greater or equal than", theta is "equal to", O is "smaller or equal than"
1) $\Omega(n^2)$ is **not correct**, because $n \log_{10} n$ grows slower than n^2
2) $\Omega(n)$ is **correct** since $n \log_{10} n$ grows faster than n
3) $\Omega(n \log_{10} n)$ is **correct**, since it is equal to the function we have
4) $\Theta(n \log_{10} n)$ is **correct**, since it is equal to the function we have
5) $\Theta(n \log_2 n)$ is **correct**, since base of the log does not matter, so this equal to our function
6) $O(n \log_{10} n)$ is **correct**, since it is equal to our function
7) $O(n)$ is **not correct**, since n grows slower than $n \log_{10} n$, so it is not greater or equal to $f(n)$
8) $O(n^2)$ is **correct**, since n^2 grows faster than $n \log_{10} n$

Quiz 2

Which iterator type allows both forward and backward traversal but not random access?

- ○ Forward Iterator
- ◉ Bidirectional Iterator
- ○ Contiguous Iterator
- ○ Random Access Iterator

Forward iterator allows only forward traversal, and contiguous and random access iterators both provide random access, so the answer is **bidirectional iterator**

Suppose that in **multiplicative** approach of dynamic array implementation we grow the capacity by a factor of 1.5 instead of 2. What would be the impact on the performance?

- ○ No difference in performance
- ◉ Lower memory wastage but higher amortized insertion cost
- ○ Higher memory wastage but lower amortized insertion cost
- ○ Insertions will always be O(1)

Since we allocate less memory after each rehash, less memory could remain unused, but at the same time we will ran out of free space more frequently, therefore a higher amortized insertion cost

A sorting algorithm is considered **stable** if:

- ○ It sorts elements in O(n log n) time
- ○ It does not use extra memory
- ◉ It preserves the relative order of equal elements
- ○ It always sorts in ascending order

This is just the definition

Suppose we want sort an array of objects, that are easy to compare, but really computationally heavy to swap. Which of this 3 quadratic sorting algorithms would be the most efficient in this case?

- ○ Bubble sort
- ◉ Selection sort
- ○ Insertion sort

Both bubble and insertion sorts do $O(n^2)$ swaps in the worst case, but selection sort always does $O(n)$ swaps.

Which of the following lines would sort vector *vec* of *Student* objects by GPA in descending order?

```
struct Student { std::string name; double gpa; };
```

- ○ std::sort(vec.begin(), vec.end(), [](Student a, Student b) { return a.gpa < b.gpa; });
- ◉ std::sort(vec.begin(), vec.end(), [](const Student &a, const Student &b) { return a.gpa > b.gpa; });
- ○ std::sort(vec.begin(), vec.end());
- ○ std::sort(vec.begin(), vec.end(), std::greater<>());

Last 2 options are not correct, since default comparators we will be sorting students by the first value, in particular by their name. Among first 2 options the first one will sort them in ascending order, so the second option is correct.

Which data structure is commonly used to implement a priority queue?

- ○ Linked List
- ◉ Binary Heap
- ○ Dictionary
- ○ Stack

In priority queue we want to be able to insert values, get the smallest element, and to be able to delete it. All this operations are exactly what binary heap provides.

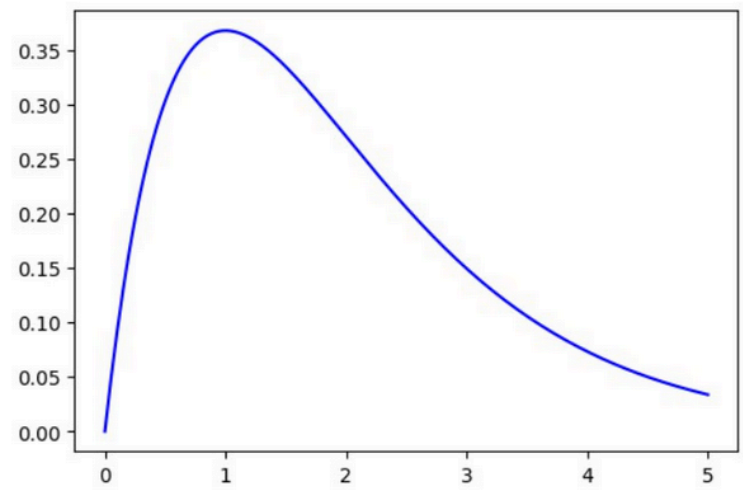
What is the average time complexity for finding the k-th smallest element using D&C approach?

- ◉ O(n)
- ○ O(nlogn)
- ○ O(nlogk)
- ○ O(n^2)

The algorithm for that is similar to quick sort, but instead of recursively going into both parts we go into only to the side that contains k-th element, so the average time complexity is linear

Quiz 3

Choose all correct facts about this function



- [ ] It's discrete function
- [x] It's continuous function
- [x] It's unimodal function
- [ ] It's monotonous function

This is not a discrete function, since we see that it is defined for non integer values, but it is a continuous function. This function has only 1 extremum, therefore it is unimodal. And lastly, it is not monotonous, since in increases, and then decreases

Given the following program what would be it's output?

```
a = [1, 1, 3, 4, 6, 7, 7, 7, 10, 11]
L = -1
R = 10
while (R - L > 1):
  M = (L + R) // 2
  if (a[M] >= 7):
    R = M
  else:
    L = M
print(L, R)
```

- ○ 3 4
- ◉ 4 5
- ○ 5 7
- ○ 6 7
- ○ 7 8
- ○ 7 9

In this implementation of binary search we are moving right boundary if the element is greater or equal to 7. Therefore in the end R would be placed on the first instance of 7, which

has the index 5. And the condition for stopping the loop is that difference between L and R is not greater then 1. Therefore L would be equal to 5 - 1 = 4. Answer is **4 5**

You have a sorted array of size n and q queries to check whether a given element exists in the array. What would be the time complexity of answering all the queries?

- ○ O(n log n + q log q)
- ○ O(n log n + n log q)
- ◉ O(q log n)
- ○ O(n log q)
- ○ O(n + q)

Array is sorted so we can use binary search for finding each element. One binary search works in $O(\log n)$ time, and since we run it 1 time for each of the q queries, the time complexity is $O(q \log n)$

What is the time complexity for finding such x that: f(x) = 0 using continuous binary search?
f - monotonous function, f(a) = -5, f(b) = 5
f can be computed in O(1) time
required precision is |x - x*| < eps

- ○ O(log n)
- ◉ O(log(b-a) - log(eps))
- ○ O(log(b - a))
- ○ O(log(b - a) + log(eps))

We are doing binary search while $r - l > \varepsilon$ to get the required precision. And since we are decreasing the distance between r and l by a factor 2 on each iteration, we will have to do $\log\left(\frac{b-a}{\varepsilon}\right)$ operations, which by properties of logarithm is equal to $\mathbf{\log(b-a) - \log(\varepsilon)}$

Quiz 4

What is the size of SinglyLinkedList<int64> and SinglyLinkedListNode<int64> in bytes, where n is the length of the list? (assuming 64-bit architecture)

- ○ 16; 16
- ◉ 16n; 16
- ○ 16n; 8
- ○ 16; 8
- ○ 8n; 16
- ○ 8; 16

int64 occupies 8 bytes, and in `SinglyLinkedListNode` we have to store 2 of them, 1 for the value, and 1 for the pointer to the next node. So the size of `SinglyLinkedListNode` would be 16 bytes. And therefore the size of `SinglyLinkedList` would be $16 \cdot n$ bytes

Which STL structures can be used to implement a stack?

- [x] std::deque
- [x] std::vector
- [ ] std::queue
- [x] std::list

Deque can be used to implement the stack, since it has both `push_back` and `pop_back` operations. Same thing with vector. Queue however is not suitable, since it does not allow `pop_back` operations. And finally list is suitable here, because again, we can both add and delete elements to the back.

A stack is used to check if a sequence of k types of brackets is a correct bracket sequence (every bracket is given by its type - a number between 1 and k - and if it's opening or closing). What is the time complexity of the algorithm?

- ( ) O(n + k)
- (•) O(n)
- ( ) O(k)
- ( ) O(nk)

The algorithm for doing that is iterating over the sequence, and maintaining a stack with opened brackets. Since operations with stack are $O(1)$ the time complexity would be just $O(n)$, because of iterating over the sequence.

Consider some array of 5 positive numbers. Let's call a subsegment "good" if its sum is no more than 10. Let f(L) be the number of "good" segments of the form [L; R]. Which of the following are valid values of f(L)?

- [x] [0, 0, 0, 0, 0]
- [x] [5, 4, 3, 2, 1]
- [x] [1, 2, 3, 2, 1]
- [ ] [1, 3, 2, 1, 2]
- [ ] [1, 3, 3, 1, 1]

Option $[1, 3, 2, 1, 2]$ is definitely invalid because for the last position of L there exists only 1 segment. Option $[1, 3, 3, 1, 1]$ is invalid because after moving from $L = 3$ to $L = 4$ value of $f(L)$ could not have decreased more then by 1. That is because if we had some valid segment $[L; R]$, then segment $[L + 1; R]$ would also be valid since we have decreased the sum. So the only segment that we lose is when $L = R$.
All other options satisfy this criteria so they are valid.

A fully persistent stack is initially empty. How many nodes are there after 3 push and 2 pop operations?

3

Persistent stack creates new node after each push, and doesn't change the amount of nodes during pop. So the answer is 3

Quiz 5

Which operations with std::set<int> X can be performed in **worst-case** O(log n)?

- [ ] union(X, Y)
- [ ] get kth order statistic (kth element in ascending order)
- [x] insert element
- [x] check if X contains x
- [ ] find the number of elements less than or equal to m

Insert and contains are $O(\log n)$. All other operations are not doable in this time complexity

Consider a function H(K). What conditions should H(K) **always** satisfy to be a **hash function**?

- [ ] H(K1) = H(K2) => K1 = K2
- [x] H(K1) != H(K2) => K1 != K2
- [ ] K1 != K2 => H(K1) != H(K2)
- [x] K1 = K2 => H(K1) = H(K2)

The main issue with hashes is that there might be collisions, so that for different values K_1 and K_2, we have same hashes. So option 3 is incorrect. Also, from this it follows that option 1 is incorrect because by having the same hashes we can not deduct that the values are the same. Both other options are correct because hash function is deterministic, so for same values it would always produce the same result.

Which of the following is a collision for this hash function?

```
int N = 97, p = 3;

int hash(std::string key) {
   int hash = 0;
   int p_pow = 1;
   for (auto c : key) {
      hash = (hash + (c - 'a') * p_pow) % N;
      p_pow = (p_pow * p) % N;
   }
   return hash;
}
```

- ○ 'aq', 'aaaq'
- ◉ 'aba', 'abaa'
- ○ 'baab', 'bab'
- ○ None of the above

Carefully looking into the code, we see that for each character a value of ‘a’ is subtracted, so if we have symbol ‘a’ in our string it will not be significant in the final value of the hash. So both strings in the second option will have the same hash, cause they have character ‘b’ on the same position

A hash table contains 5000 elements and has capacity 2^15 = 32768. What is the approximate **load factor** of this hash table?

- ○ 0.0004
- ○ 0.0032
- ◉ 0.1526
- ○ 6.5536
- ○ 333.33
- ○ 2500

$\frac{5000}{32768} \approx 0.1526$

Consider a hash table with open addressing which is rebuilt only when its load factor reaches 1. The hash table is initially empty. What's the **average** time complexity of n insert operations?

- ○ O(n sqrt(n))
- ○ O(n^2)
- ○ O(n)
- ◉ O(n log n)

Let's say we have n elements and the capacity is now 2n. We are making n insertions. On average first one will take $\frac{2n}{n}$, second will take $\frac{2n}{n-1}$ and so on. Overall we will have $2n(\frac{1}{n} + \frac{1}{n-1} + ... + \frac{1}{1}) \leq 2n \log n$, thus, $O(n \log n)$

An array a is fixed. You are given queries: for a given segment [L...R], calculate f(a[L], a[L+1], ..., a[R]). For which f can this problem be solved using the prefix approach? (choose all that apply)

- ☑ product
- ☐ gcd
- ☑ sum
- ☐ min

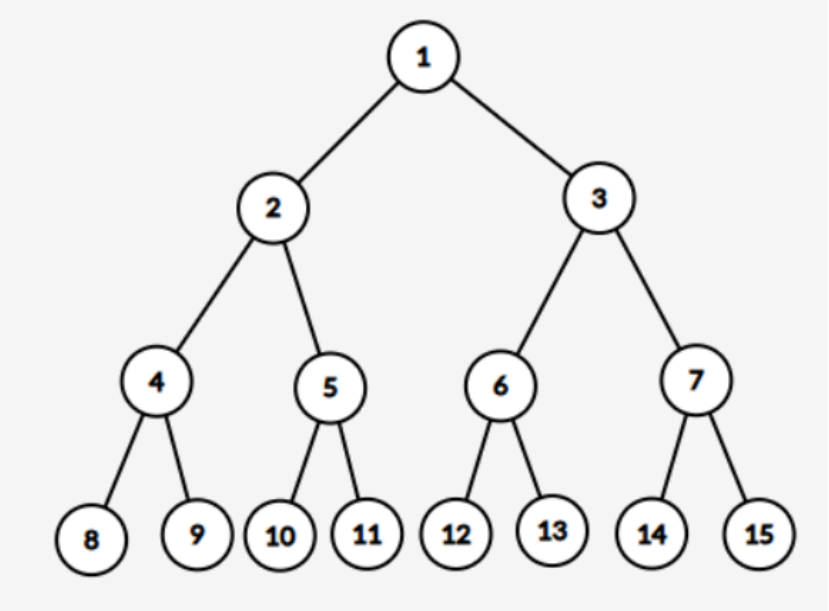
Operations that have an inverse operation are suitable here, therefore product and sum.

Quiz 6

There is a segment tree for the sum, built on a 8-element array.

You are going to change the value of the element on the position 2 **in the array** (0-indexed).

What nodes will be updated?

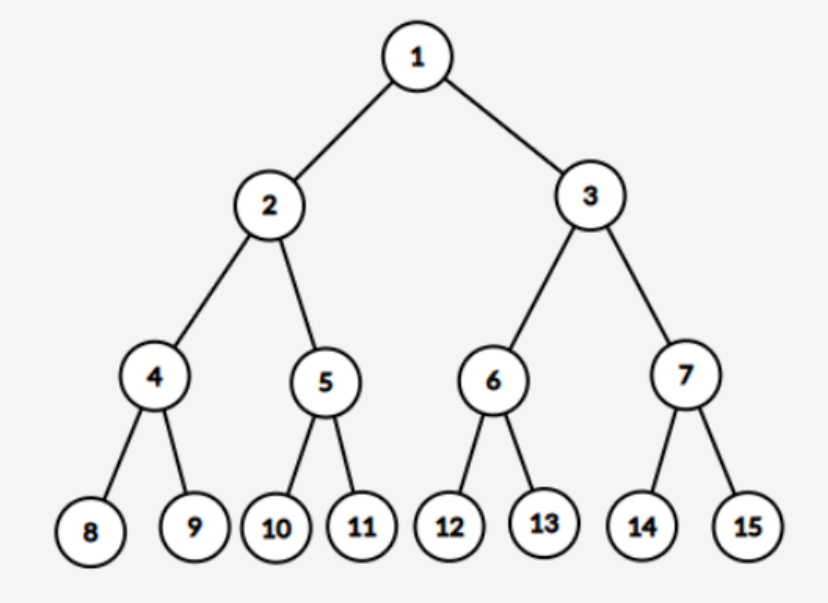


- (●) 1, 2, 5, 10
- ( ) 1, 3, 6, 13
- ( ) 1, 3, 7, 14
- ( ) 1, 2, 4, 9
- ( ) 1, 2, 4, 8

We will update all the nodes on the path from vertex 10 to the root, therefore 1, 2, 5, 10

There a is segment tree for the minimum, built on an 8-element array. *

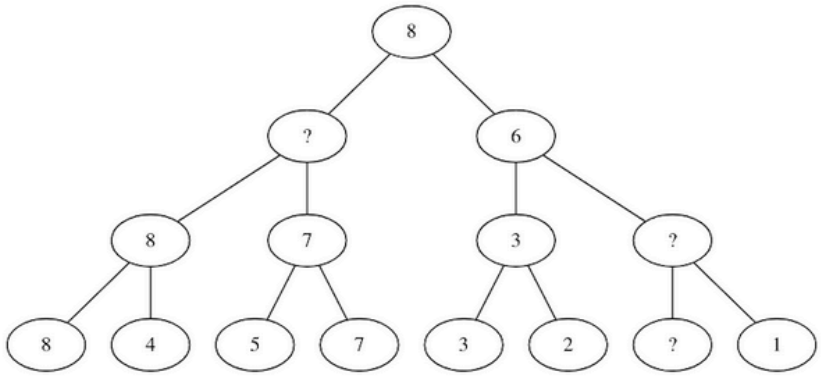
You are asking minimum on the segment [1..5] (0-indexed).
What nodes will be used for the result?



- (●) 5, 6, 9
- ( ) 5, 9, 14
- ( ) 6, 7, 10
- ( ) 2, 3
- ( ) 4, 5, 6
- ( ) 7, 11, 12
- ( ) 2, 6
- ( ) 6, 8, 11

All the nodes that are entirely inside of the query segment will be used. In this case it will be node 9, 5, 6.

On the picture, there is a segment tree for the maximum. What are the values in the missing nodes?



- ○ 8, 5, 3
- ○ 7, 1, 6
- ◉ 8, 6, 6
- ○ 7, 1, 1
- ○ 8, 4, 4

The first missing value is 8, since value in the node is maximum of its children, so max(7, 8) = 8. Next up we have 2 nodes on the right. The upper one must contain value 6, since it's parent has value 6, which should be the maximum of both children, but left child has value 3. And lastly the lower node should also contain value 6 following the same logic.

Given an array a of length n and a function *f*, we need to answer the following queries:
- calculate *f* on a[L...R];
- set a[i] = x.

For which *f* can we answer these queries in O(log n) using a segment tree?

- ☑ f(a[L...R]) = a_L * a_{L+1} * ... * a_R
- ☐ f(a[L...R], x) = number of L <= i <= R such that a_i >= x
- ☑ f(a[L...R], x) = smallest L <= i <= R such that a_i >= x
- ☑ f(a[L...R]) = the number of L <= i <= R such that a_i = 0
- ☐ f(a[L...R]) = a_L ^ (a_{L+1} ^ (... ^ (a_R)...)) % M

Product is trivial, smallest i such that $a_i \geq x$ is done having segment tree for maximum and using the idea of descending. The number of zeros is done using segment tree for sum, and changing 0 values to 1, and all others to 0.

A persistent segment tree is built on an array of length 8. How many vertices are there after 3 modifications?

Review

- ○ 11
- ○ 18
- ○ 15
- ◉ 27
- ○ 20
- ○ 22

Initially there are $8 + 4 + 2 + 1 = 15$ vertices, on each update we add new nodes on the way from the root to the leaf, which is 4. So after 3 updates we will add 12 vertices resulting in 27 vertices in total.

Quiz 7

Consider a tree T with **n** vertices.
How many **edges** does T have?
Let's call a sequence (v_1, v_2, ..., v_k), k ≥ 2 a **path** if for any 1 ≤ i < k, (v_i, v_{i+1}) is an edge in T. Paths (v_1, v_2, ..., v_k) and (v_k, ..., v_2, v_1) are considered to be the same.
How many **paths** does T have?

- ○ n-1; n^2
- ○ 2n; n(n-1)/2
- ○ n; n^2
- ○ 2n; 2^n
- ○ n; n(n-1)/2
- ◉ n-1; n(n-1)/2
- ○ n(n-1)/2; n^2

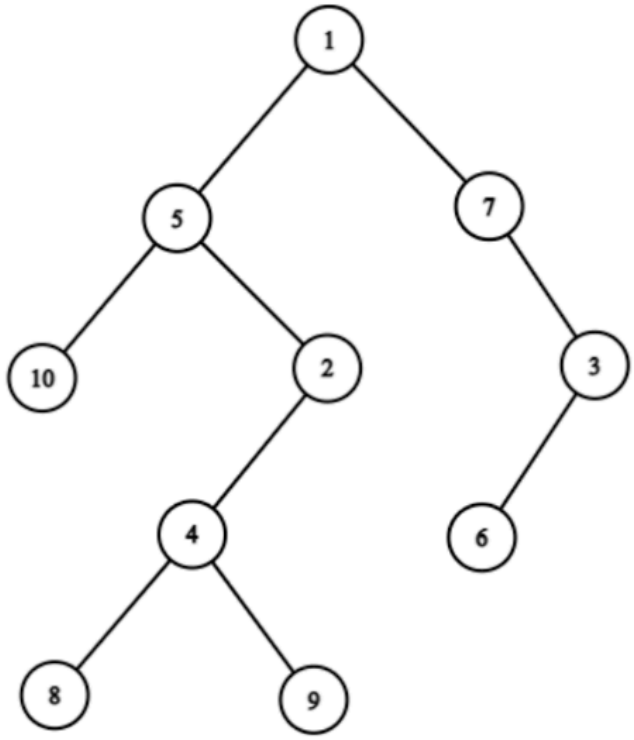
Tree is a connected graph without cycles, so there should be at least $n-1$ edge to make it connected, and there can be no more than $n-1$ edge, since otherwise there would be cycles. There is exactly 1 path between each pair of vertices, so the number of paths is the number of pairs of vertices we can choose. This is simply $\binom{n}{2} = \frac{n \cdot (n-1)}{2}$
Answer is $n-1$; $\frac{n \cdot (n-1)}{2}$

Review

You are staying in **node 2**.

What is the **next node** in the traverse order?

1) LVR
2) VLR
3) LRV



- ○ LVR: 4, VLR: 5, LRV: 7
- ○ LVR: 1, VLR: 5, LRV: 7
- ○ LVR: 1, VLR: 8, LRV: 6
- ○ LVR: 4, VLR: 1, LRV: 8
- ◉ LVR: 1, VLR: 4, LRV: 5
- ○ LVR: 5, VLR: 8, LRV: 6
- ○ LVR: 5, VLR: 4, LRV: 1
- ○ LVR: 4, VLR: 8, LRV: 5
- ○ LVR: 5, VLR: 1, LRV: 7

The order of traversal for the whole tree for all 3 cases:
1) LVR: 10, 5, 8, 4, 9, **2, 1**, 6, 3, 7
2) VLR: 1, 5, 10, **2, 4**, 8, 9, 7, 3, 6
3) LRV: 10, 8, 9, 4, **2, 5**, 6, 3, 7, 1
So the answer is 1, 4, 5

In a binary tree T of size n, let L and R be the **left** and the **right** subtrees of a vertex v respectively. Let H(L) be the height of the left subtree, S(L) be its size; same for R. Which of these conditions imply that T is balanced?

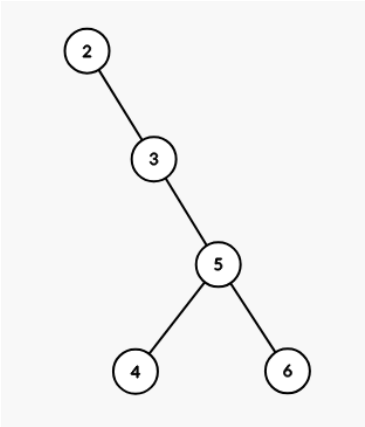
- [x] for all v, |S(L) - S(R)| < 1
- [ ] T has at least n/2 leaves
- [x] for all v, |H(L) - H(R)| < 2025
- [x] for all v, |H(L) - H(R)| < 1
- [x] for all v, |S(L) - S(R)| < 2025

All options with inequalities are correct, since tree is balanced if the height is $O(\log n)$, and it can be shown that if for every vertex difference in size, or height between children does not exceed some constant C, in our case 1 or 2025, then the height of such tree is $O(\log n)$

A BST is initially empty. Values are inserted in the following order: 2, 3, 5, 4, 6. What is the resulting **height** of the BST (height defined as the number of edges on the longest path from root to leaf)?

- ( ) 1
- ( ) 2
- (•) 3
- ( ) 4
- ( ) 5

The tree would look like:



So it can be seen that the height is 3.

A BST is initially empty. n values are chosen randomly and uniformly from the interval [0; M], where M is sufficiently greater than n, and inserted into the BST. What is the expected value of the resulting height?

- ○ O(1)
- ◉ O(log n)
- ○ O(sqrt(n))
- ○ O(n)

Inserting random values that are uniformly distributed would result in expected $O(\log(n))$ height, since each inserted element is equally likely to split the remaining elements into subgroups that are kind of balanced. This resembles the quick sort where we randomly pick a pivot element.

Let `std::set s = {2, 1, 4, 5, 2, 3}'.

What are the results of

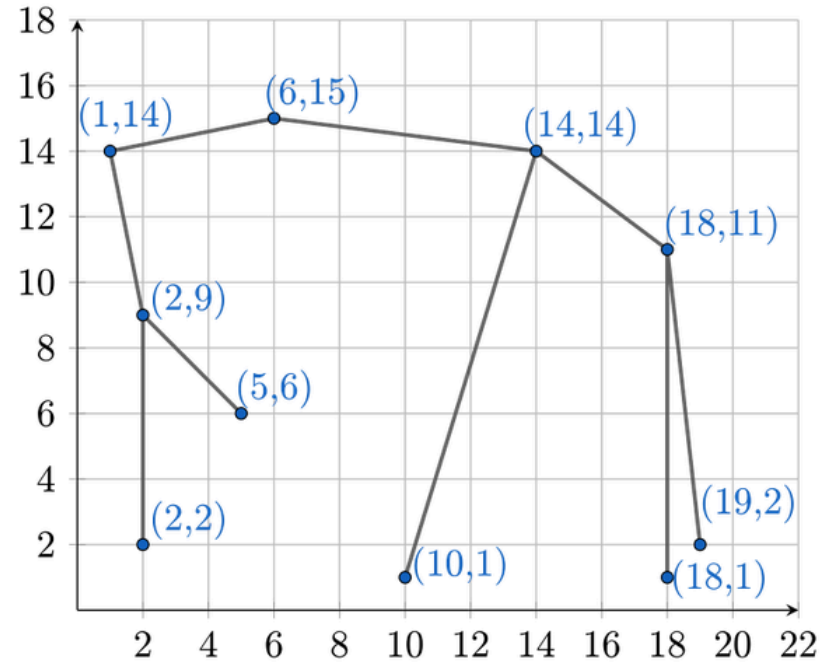
1) `lower_bound(3) - begin()`;
2) `upper_bound(4) - begin()`?

- ○ 2, 3
- ◉ 2, 4
- ○ 2, 5
- ○ 3, 3
- ○ 3, 4
- ○ 3, 5
- ○ 4, 3
- ○ 4, 4
- ○ 4, 5

Set would look like $\{1, 2, 3, 4, 5\}$. Lower bound would find element 3, since it is the first element not less than 3. Begin points at the first element. So the result of subtraction is 2 Upper bound would find element 5, since it is the first element greater than 4. And again, since begin points at the first element the result would be 4.

You are given the following treap T. It is split by key k=4 into trees L and R. What is the depth of L?

Review

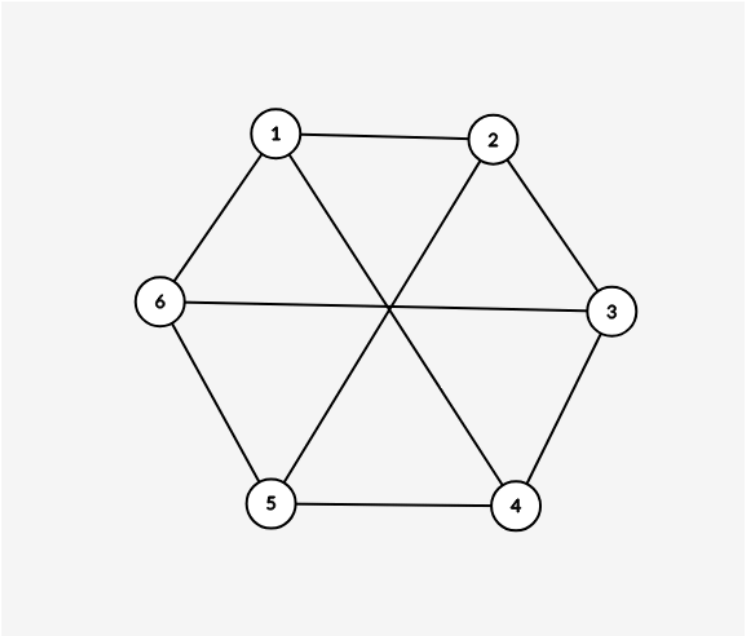


- 0
- 1
- 2
- 3

Splitting by $k = 4$ we get 3 elements in tree L: (1, 14), (2, 9), (2, 2). The height of the tree would be 2.

Quiz 8

The graph below is ... (choose all correct statements)



- [x] bipartite
- [x] connected
- [ ] acyclic
- [ ] a tree
- [x] regular

This graph **is** clearly connected, since it has only 1 connectivity component.
This graph **is** regular, since each vertex has an equal amount of edges.
This graph **is** bipartite, since we can color vertices from 1 to 6 with alternating colors, and no 2 adjacent vertices will have the same color.
This graph **is not** acyclic, since there are clearly cycles in it, for example 1-4-5-6-1
This graph **is not** a tree, since it is not acyclic.

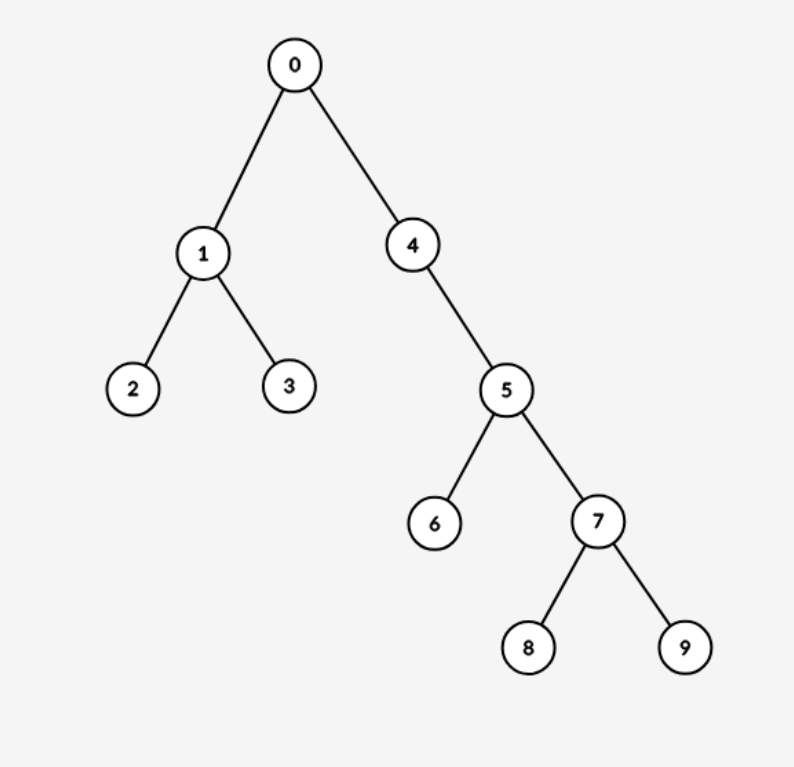
What's the maximum possible number of bridges in a graph on 10 vertices?
Articulation points?

- ○ 10; 10
- ◉ 9; 8
- ○ 9; 9
- ○ 10; 8
- ○ 8; 55

If we remove the bridges from the graph, it will split into several connectivity components, the maximum we can get is 10, because there are 10 vertices, so the maximum amount of bridges would be 9. An example of such graph is path graph.

For articulation points same example with path graph works, this way we get 8 articulation points (all vertices except endpoints)

How many centroids does the following tree have? Write the answer as a single positive integer



2

There are 2 centroids: 4 and 5, since for each of them if we remove it, the remaining trees would have not more than $\lfloor \frac{n}{2} \rfloor$ vertices