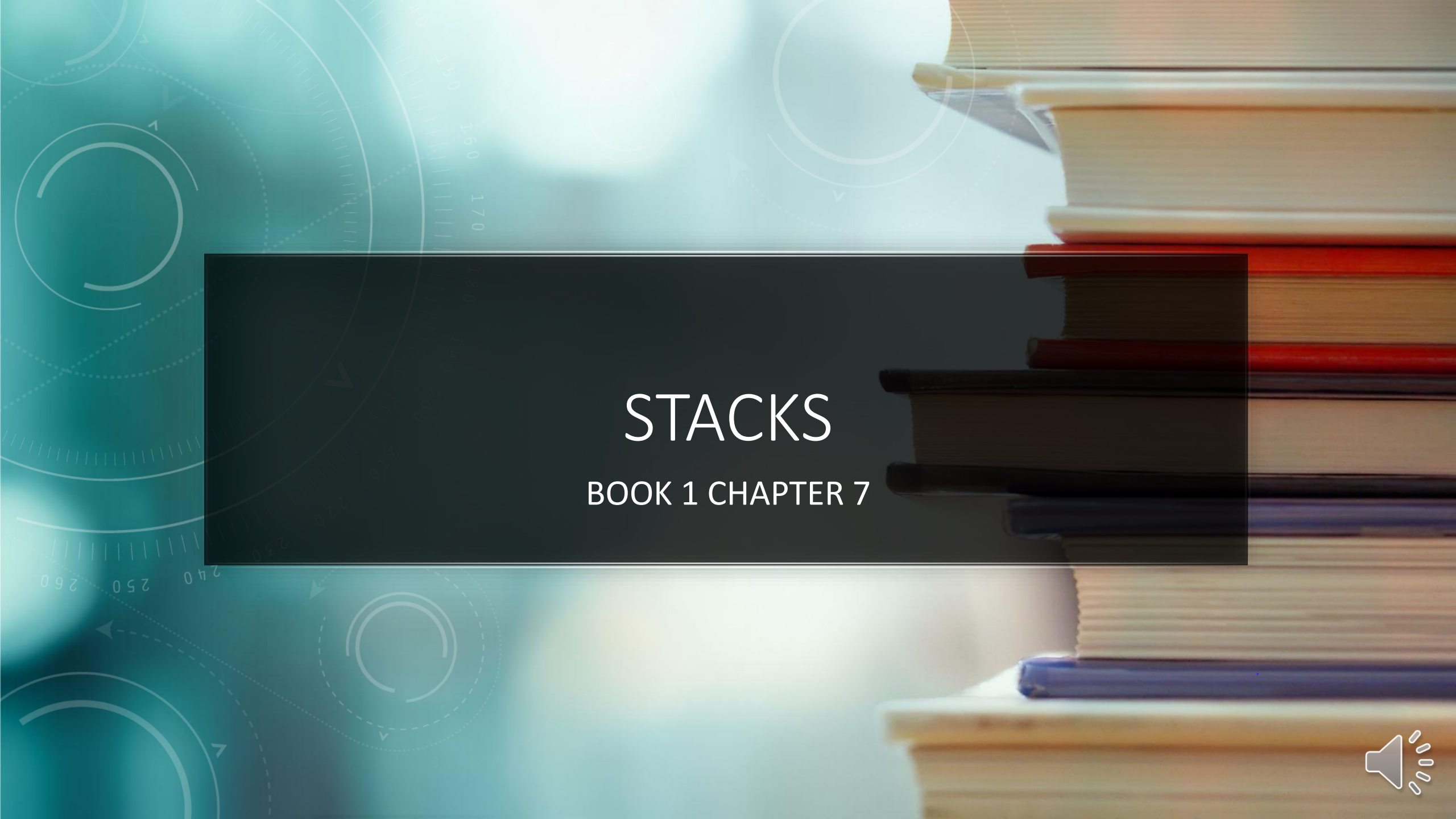
CS-218 DATA STRUCTURES AND ALGORITHMS

LECTURE 15

BY UROOJ AINUDDIN

STACKS

BOOK 1 CHAPTER 7



IN THE LAST LECTURE…

We discussed the reasons behind preference of postfix expressions for computing machines.

We traced through the algorithm for postfix evaluation.

We traced through the algorithm for conversion from infix to postfix expression.

We investigated both time and space complexities of the algorithms discussed.

QUICKSORT

SORTING WITH STACKS

ABOUT THE ALGORITHM

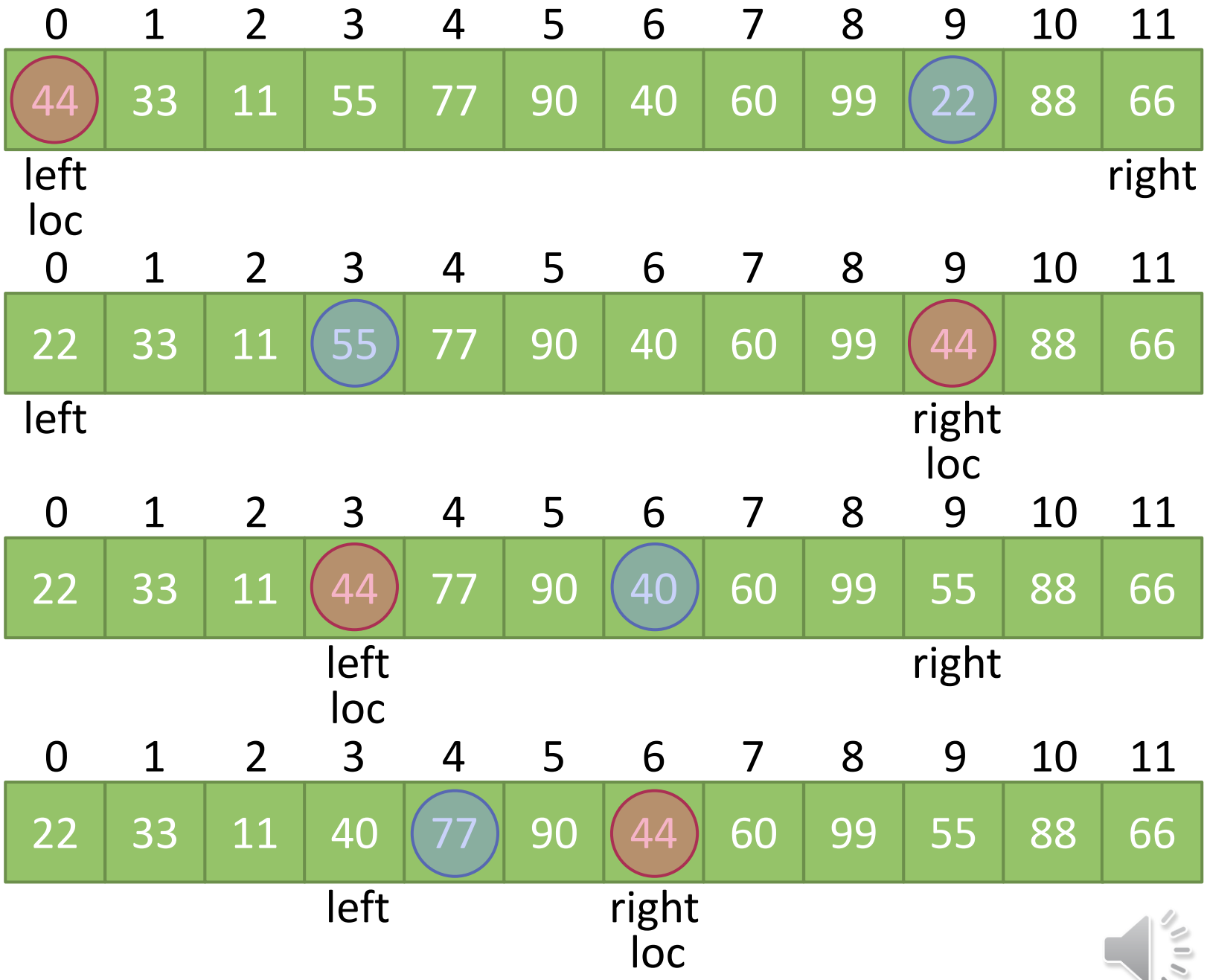
- The quicksort algorithm uses the **divide and conquer** strategy.
- In divide and conquer approach, the problem in hand is divided into smaller subproblems and then each problem is solved independently.
- Since we keep on dividing the subproblems into even smaller subproblems, we eventually reach a stage where no more division is possible. These "atomic" smallest possible subproblems are solved.
- The solution of all subproblems is finally merged to obtain the solution of the original problem.
- The quicksort partitions the sequence by dividing it into two segments based on a selected **pivot** key.

ALGORITHM

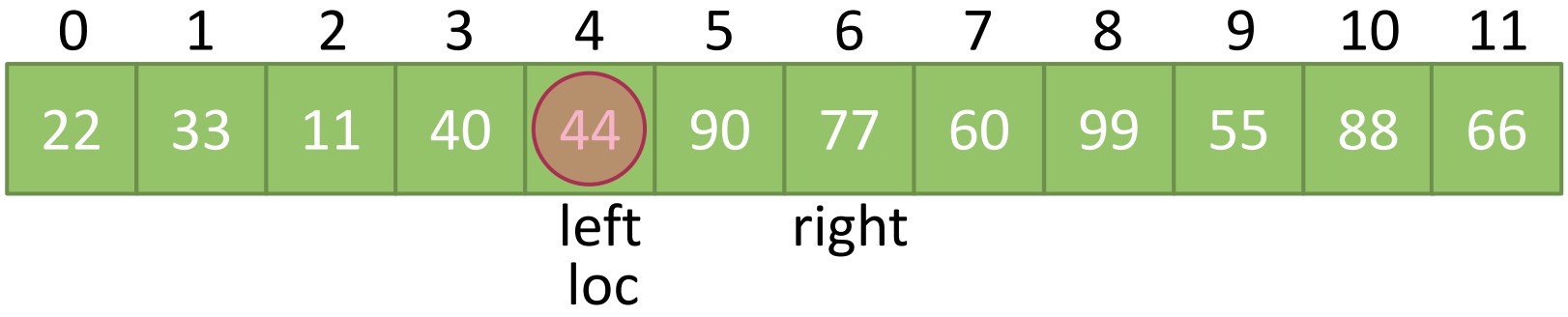
1. If the stack is empty, quit. Otherwise, pop a sequence from the stack.
2. Select the first element as the pivot, p. (The location of p is called loc, and is updated with p.)
3. Begin from the right. Scan the sequence of keys from right to loc, looking for a smaller key than p. If you find it,swap it with p. (Mark the new loc.) Let this position be the new right.
4. Begin from the left. Scan the sequence of keys from left to loc, looking for a larger key than p. If you find it, swap it with p. (Mark the new loc.) Let this position be the new left.
5. Repeat steps 3 and 4 until left and right coincide. When they do, the pivot is at its correct position. On its left is a sequence L with all values less than p. On its right is a sequence G with all values greater than p.
6. Push L onto the stack. Then push G onto the stack.
7. Go to step 1.

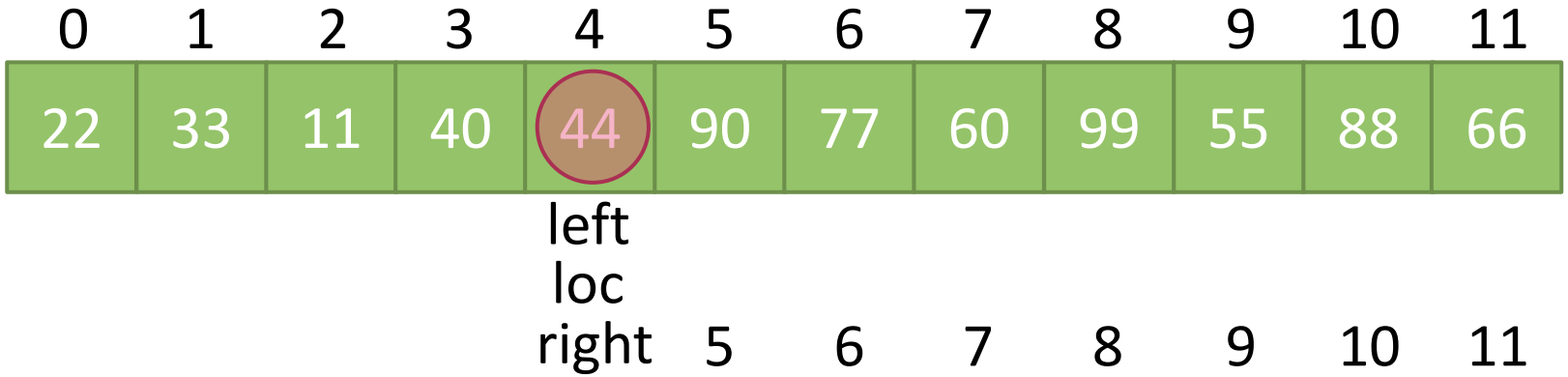
1. If the stack is empty, quit. Otherwise, pop a sequence from the stack.
2. Select the first element as the pivot, p. (The location of p is called loc, and is updated with p.)
3. Begin from the right. Scan the sequence of keys from right to loc, looking for a smaller key than p. Swap it with p. (Mark the new loc.) Let this position be the new right.
4. Begin from the left. Scan the sequence of keys from left to loc, looking for a larger key than p. Swap it with p. (Mark the new loc.) Let this position be the new left.
5. Repeat steps 3 and 4 until left and right coincide. When they do, the pivot is at its correct position. On its left is a sequence L with all values less than p. On its right is a sequence G with all values greater than p.
6. Push L onto the stack. Then push G onto the stack.
7. Go to step 1.

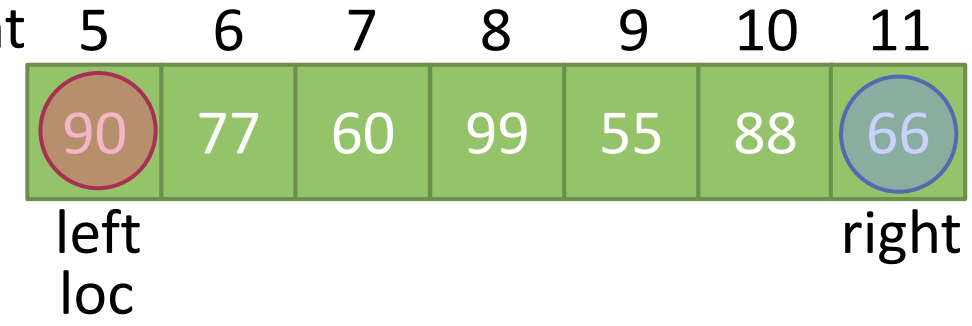
[0,11]

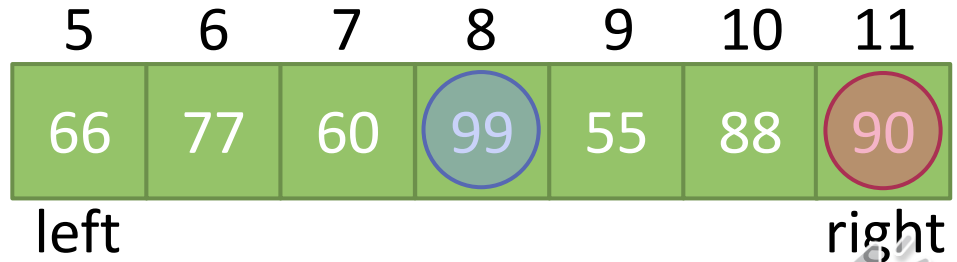


1. If the stack is empty, quit. Otherwise, pop a sequence from the stack.
2. Select the first element as the pivot, p. (The location of p is called loc, and is updated with p.)
3. Begin from the right. Scan the sequence of keys from right to loc, looking for a smaller key than p. Swap it with p. (Mark the new loc.) Let this position be the new right.
4. Begin from the left. Scan the sequence of keys from left to loc, looking for a larger key than p. Swap it with p. (Mark the new loc.) Let this position be the new left.
5. Repeat steps 3 and 4 until left and right coincide. When they do, the pivot is at its correct position. On its left is a sequence L with all values less than p. On its right is a sequence G with all values greater than p.
6. Push L onto the stack. Then push G onto the stack.
7. Go to step 1.





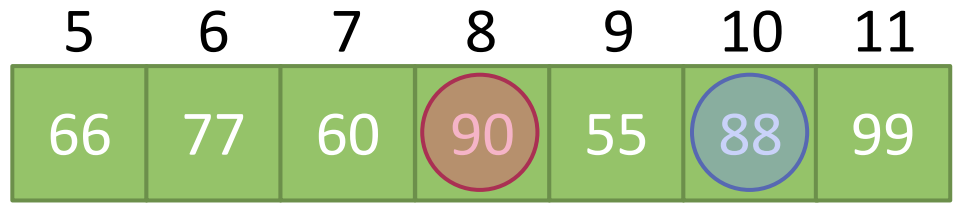


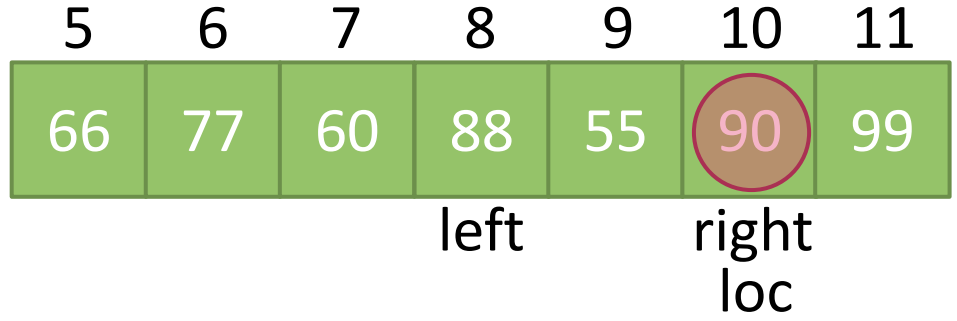


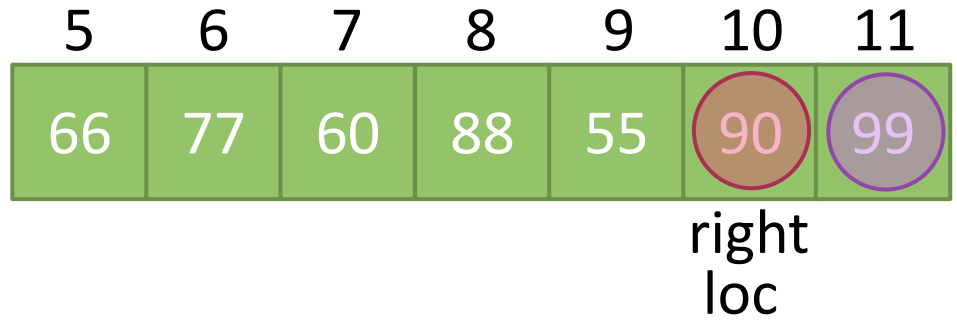
1. If the stack is empty, quit. Otherwise, pop a sequence from the stack.
2. Select the first element as the pivot, p. (The location of p is called loc, and is updated with p.)
3. Begin from the right. Scan the sequence of keys from right to loc, looking for a smaller key than p. Swap it with p. (Mark the new loc.) Let this position be the new right.
4. Begin from the left. Scan the sequence of keys from left to loc, looking for a larger key than p. Swap it with p. (Mark the new loc.) Let this position be the new left.
5. Repeat steps 3 and 4 until left and right coincide. When they do, the pivot is at its correct position. On its left is a sequence L with all values less than p. On its right is a sequence G with all values greater than p.
6. Push L onto the stack. Then push G onto the stack.
7. Go to step 1.

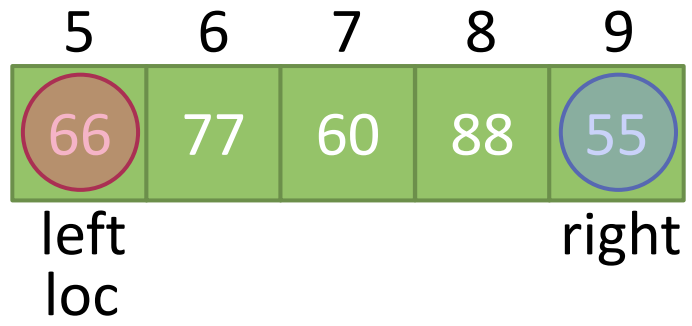
[5,9]

[0,3]

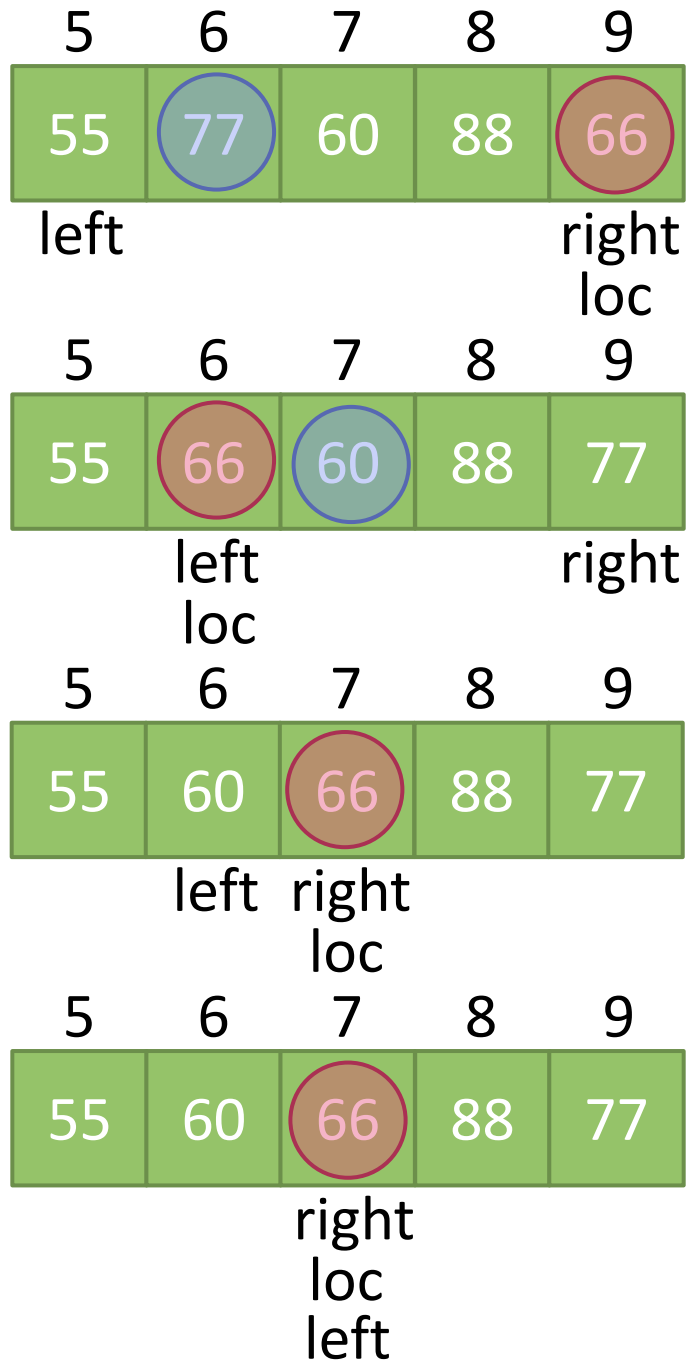




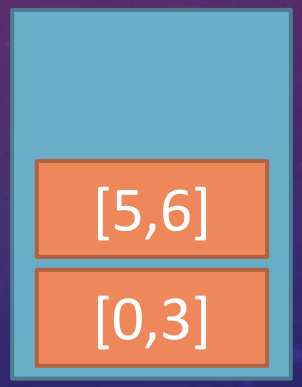


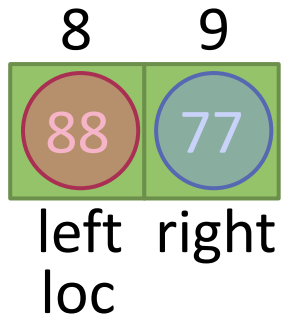


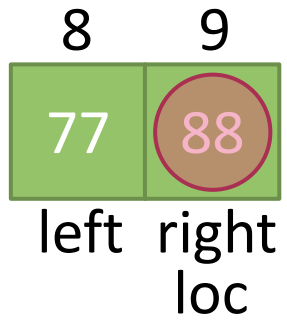
1. If the stack is empty, quit. Otherwise, pop a sequence from the stack.
2. Select the first element as the pivot, p. (The location of p is called loc, and is updated with p.)
3. Begin from the right. Scan the sequence of keys from right to loc, looking for a smaller key than p. Swap it with p. (Mark the new loc.) Let this position be the new right.
4. Begin from the left. Scan the sequence of keys from left to loc, looking for a larger key than p. Swap it with p. (Mark the new loc.) Let this position be the new left.
5. Repeat steps 3 and 4 until left and right coincide. When they do, the pivot is at its correct position. On its left is a sequence L with all values less than p. On its right is a sequence G with all values greater than p.
6. Push L onto the stack. Then push G onto the stack.
7. Go to step 1.

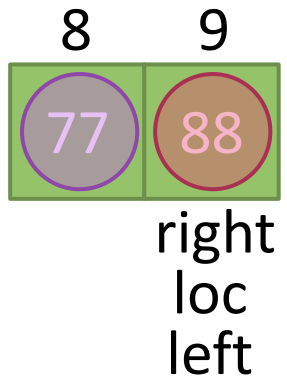


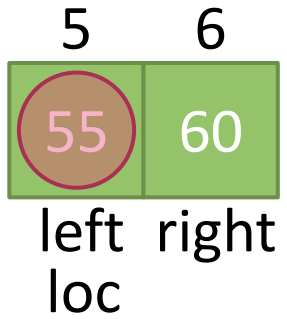
1. If the stack is empty, quit. Otherwise, pop a sequence from the stack.
2. Select the first element as the pivot, p. (The location of p is called loc, and is updated with p.)
3. Begin from the right. Scan the sequence of keys from right to loc, looking for a smaller key than p. Swap it with p. (Mark the new loc.) Let this position be the new right.
4. Begin from the left. Scan the sequence of keys from left to loc, looking for a larger key than p. Swap it with p. (Mark the new loc.) Let this position be the new left.
5. Repeat steps 3 and 4 until left and right coincide. When they do, the pivot is at its correct position. On its left is a sequence L with all values less than p. On its right is a sequence G with all values greater than p.
6. Push L onto the stack. Then push G onto the stack.
7. Go to step 1.





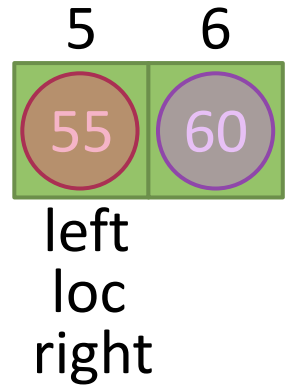


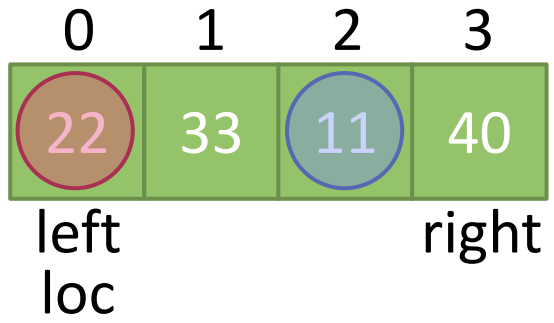


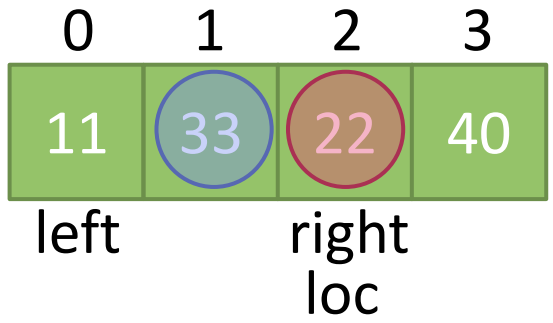


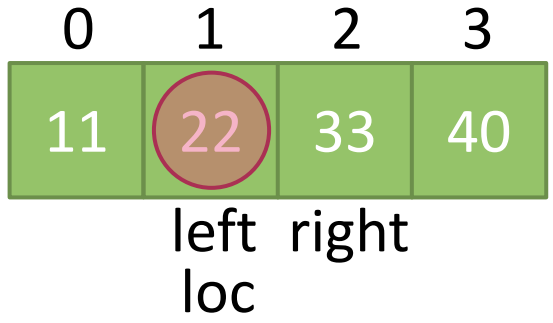
1. If the stack is empty, quit. Otherwise, pop a sequence from the stack.
2. Select the first element as the pivot, p. (The location of p is called loc, and is updated with p.)
3. Begin from the right. Scan the sequence of keys from right to loc, looking for a smaller key than p. Swap it with p. (Mark the new loc.) Let this position be the new right.
4. Begin from the left. Scan the sequence of keys from left to loc, looking for a larger key than p. Swap it with p. (Mark the new loc.) Let this position be the new left.
5. Repeat steps 3 and 4 until left and right coincide. When they do, the pivot is at its correct position. On its left is a sequence L with all values less than p. On its right is a sequence G with all values greater than p.
6. Push L onto the stack. Then push G onto the stack.
7. Go to step 1.

[0,3]



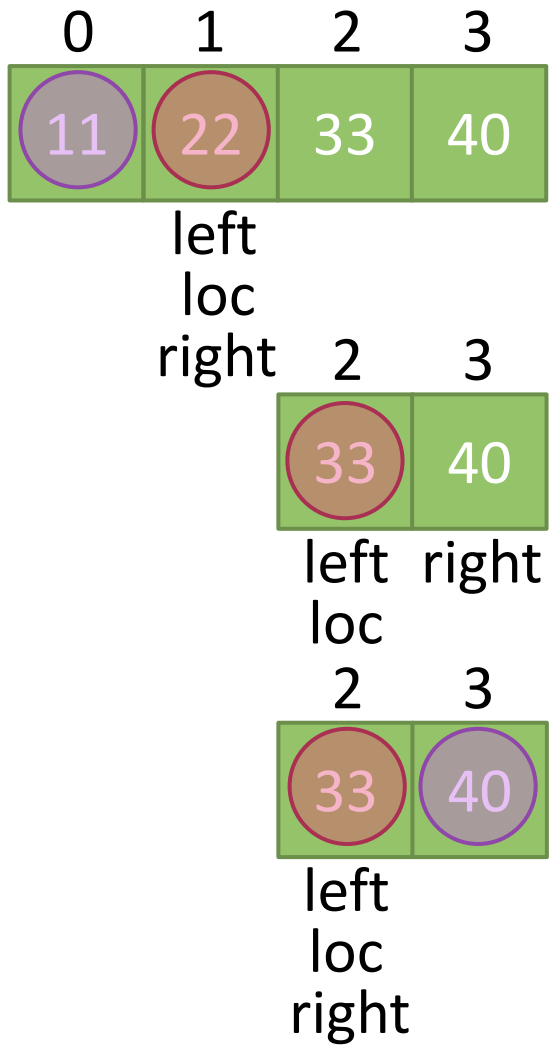


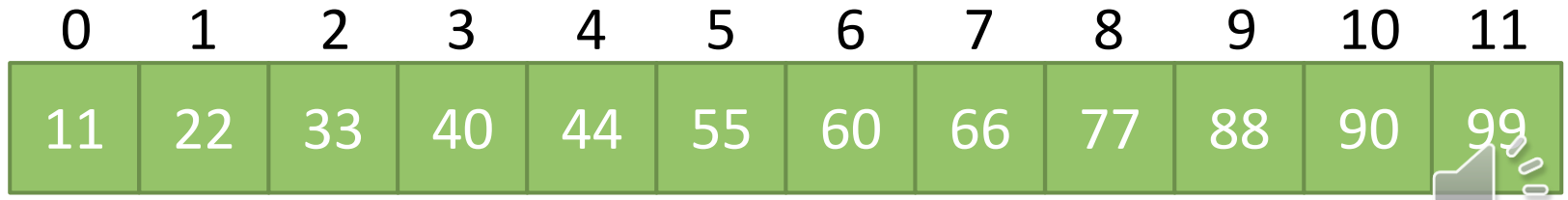




1. If the stack is empty, quit. Otherwise, pop a sequence from the stack.
2. Select the first element as the pivot, p. (The location of p is called loc, and is updated with p.)
3. Begin from the right. Scan the sequence of keys from right to loc, looking for a smaller key than p. Swap it with p. (Mark the new loc.) Let this position be the new right.
4. Begin from the left. Scan the sequence of keys from left to loc, looking for a larger key than p. Swap it with p. (Mark the new loc.) Let this position be the new left.
5. Repeat steps 3 and 4 until left and right coincide. When they do, the pivot is at its correct position. On its left is a sequence L with all values less than p. On its right is a sequence G with all values greater than p.
6. Push L onto the stack. Then push G onto the stack.
7. Go to step 1.

[2,3]



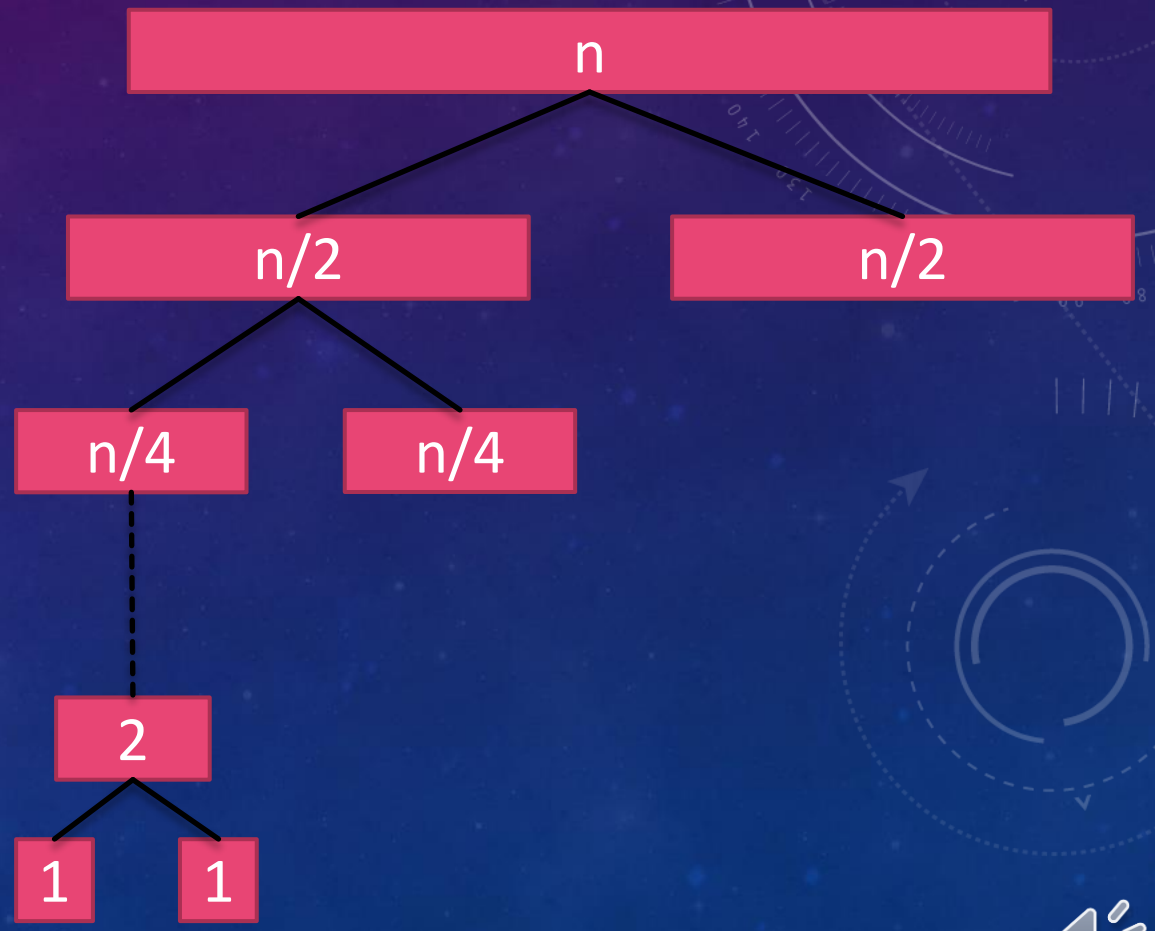


HOMEWORK

1. Run the quicksort algorithm for a sequence of 8 keys sorted in descending order.
2. Run the quicksort algorithm for a sequence of 8 keys sorted in ascending order.

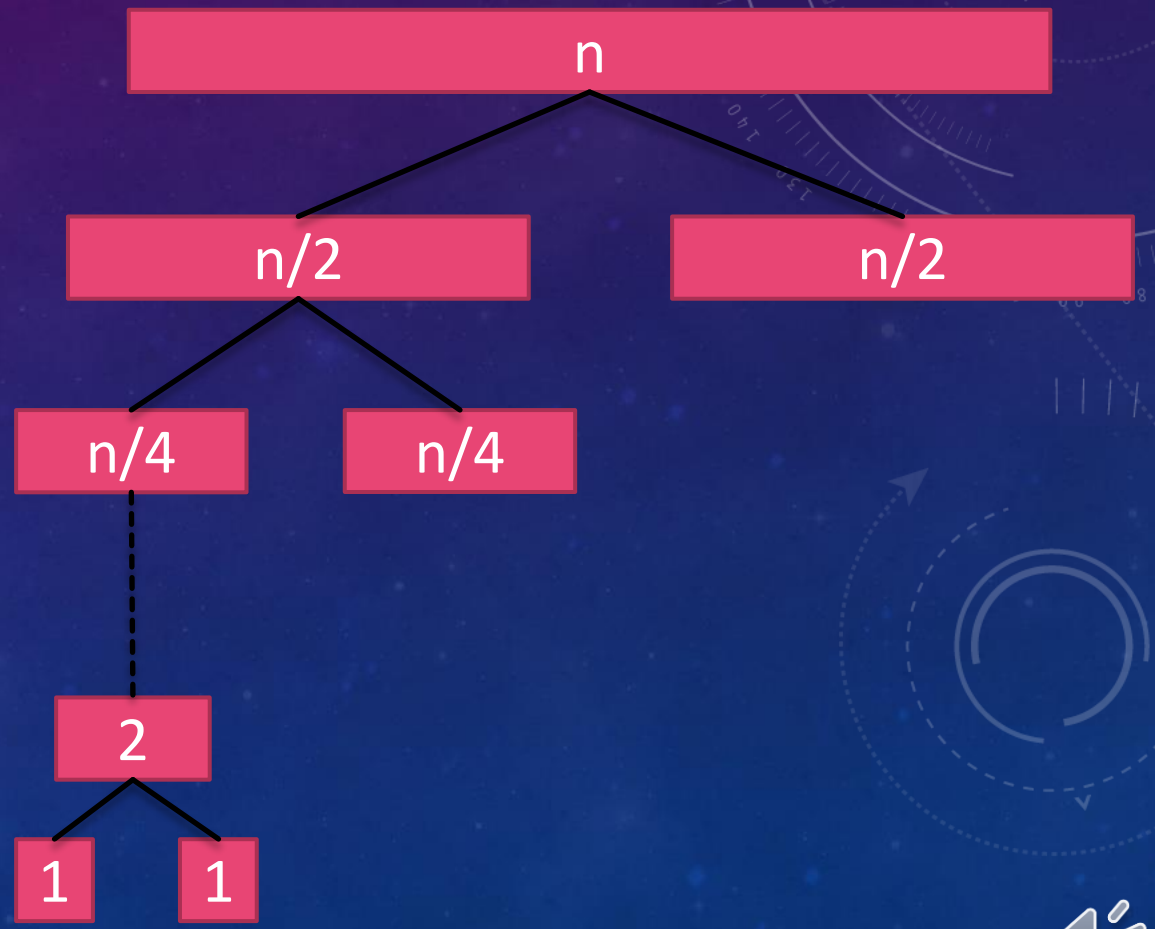
BEST CASE RUNNING TIME OF QUICKSORT

- The best case of running time of quicksort happens when the correct position of the pivot is in the middle of the n-element sequence in every pass.
- Two subsequences of roughly n/2 length break off on both sides of the pivot in the first pass.
- Two subsequences of roughly n/4 length break off on both sides of the pivot in the second pass.
- In the last pass, the two subsequences that break off on both sides have only one element.



BEST CASE RUNNING TIME OF QUICKSORT

- Let $T(n)$ be the time taken to sort n elements.
- Then the time taken to sort $n/2$ elements must be $T\left(\frac{n}{2}\right)$.
- We compare the first pivot with n elements in the sequence to place it in its correct position, which is the middle of the sequence.
- Time to sort one element is constant, so we say $T(1) = 1$.



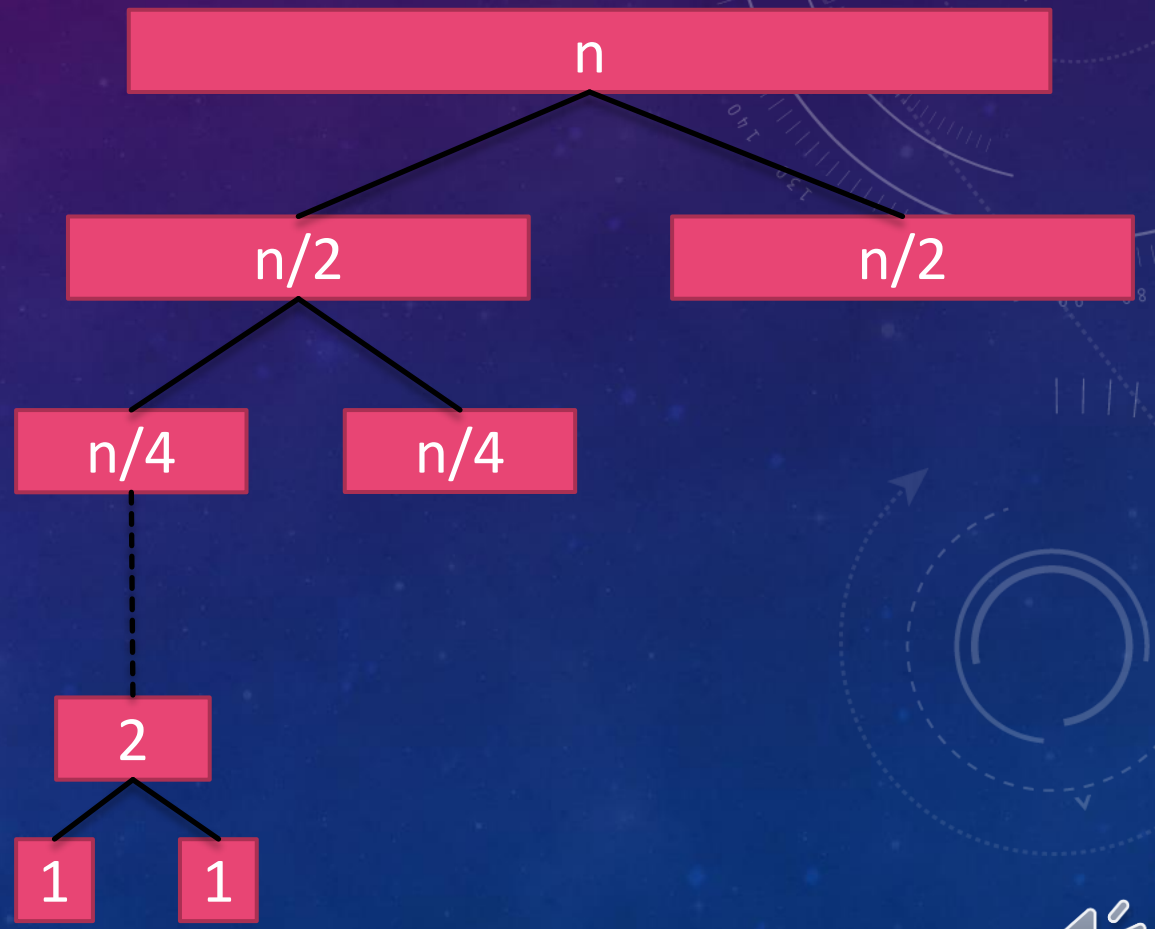
BEST CASE RUNNING TIME OF QUICKSORT

- Time to sort n elements = Number of comparisons to place the pivot in its correct position + time to sort the subsequence of values lesser than the pivot + time to sort the subsequence of values greater than the pivot.

$$T(n) = n + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right), T(1) = 1$$

$$T(n) = 2\,T\left(\frac{n}{2}\right) + n, T(1) = 1$$

- We will now solve this recurrence relation using backtracking.



BEST CASE RUNNING TIME OF QUICKSORT

$$T(n) = 2\,T\left(\frac{n}{2}\right) + n$$

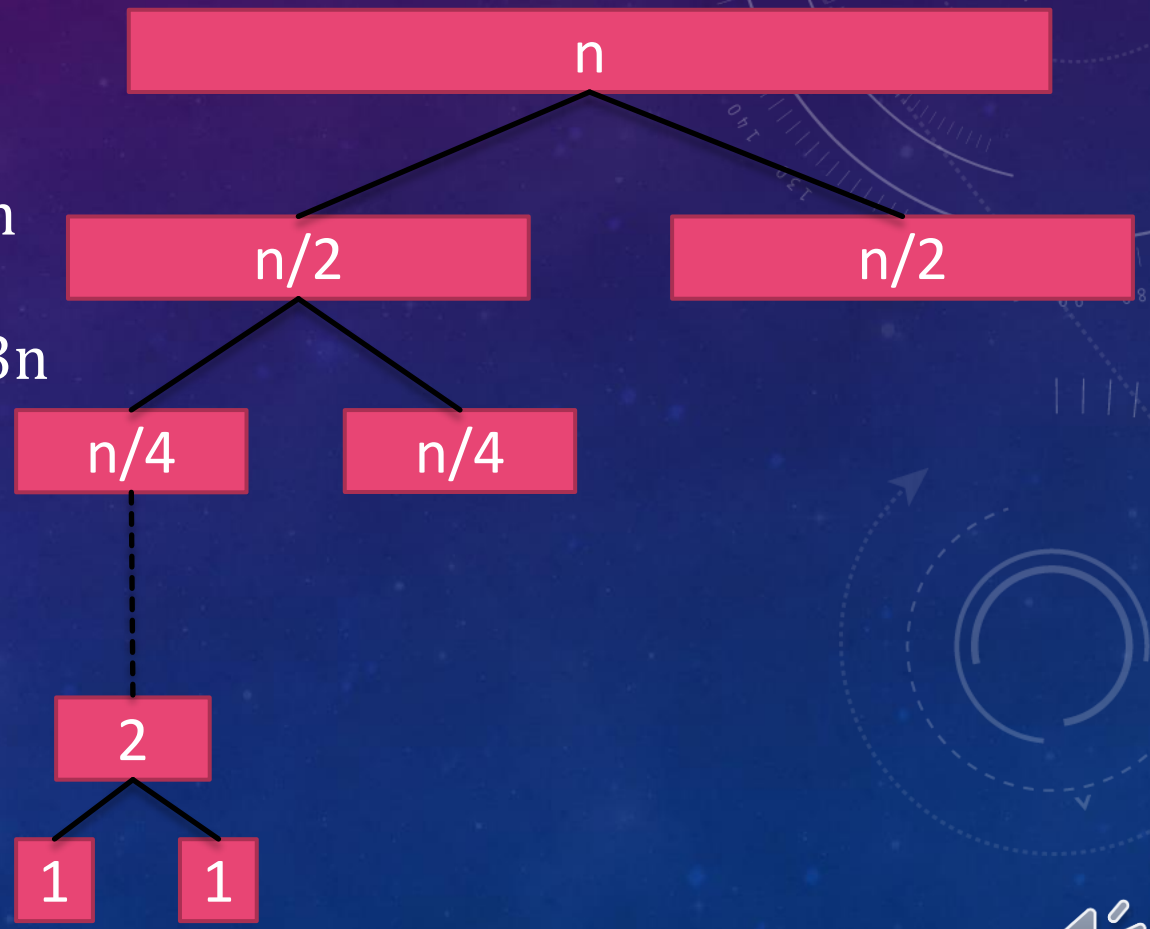
$$T(n) = 2\left[2\,T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n = 2^2\,T\left(\frac{n}{4}\right) + 2n$$

$$T(n) = 2^2\left[2\,T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n = 2^3\,T\left(\frac{n}{8}\right) + 3n$$

- Generalizing for k, we can say that:

$$T(n) = 2^k\,T\left(\frac{n}{2^k}\right) + kn$$

- We want $\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow \log n = k.$



BEST CASE RUNNING TIME OF QUICKSORT

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$$

- Putting $k = \log n$:

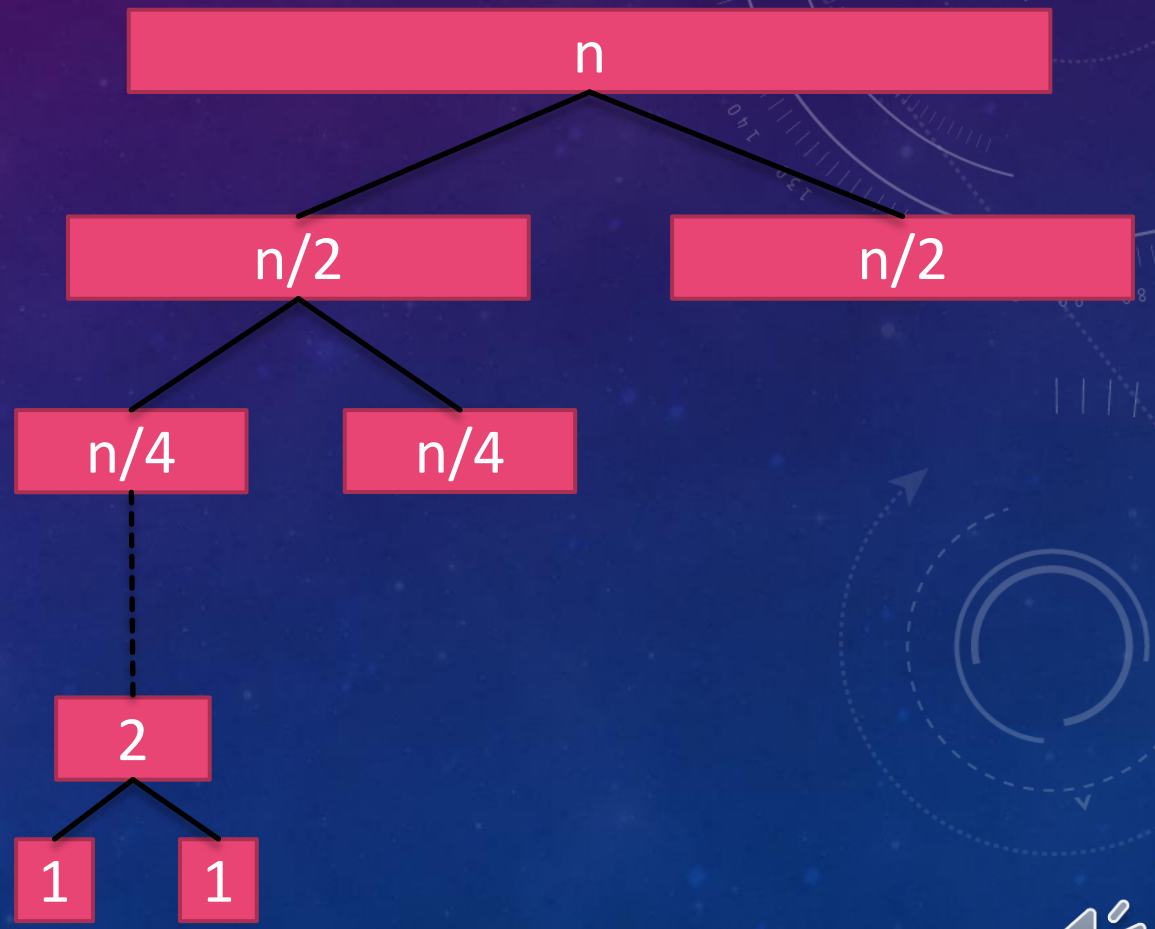
$$T(n) = n\, T\left(\frac{n}{n}\right) + (\log n)n$$

$$T(n) = n\, T(1) + n \log n$$

- But $T(1) = 1$.

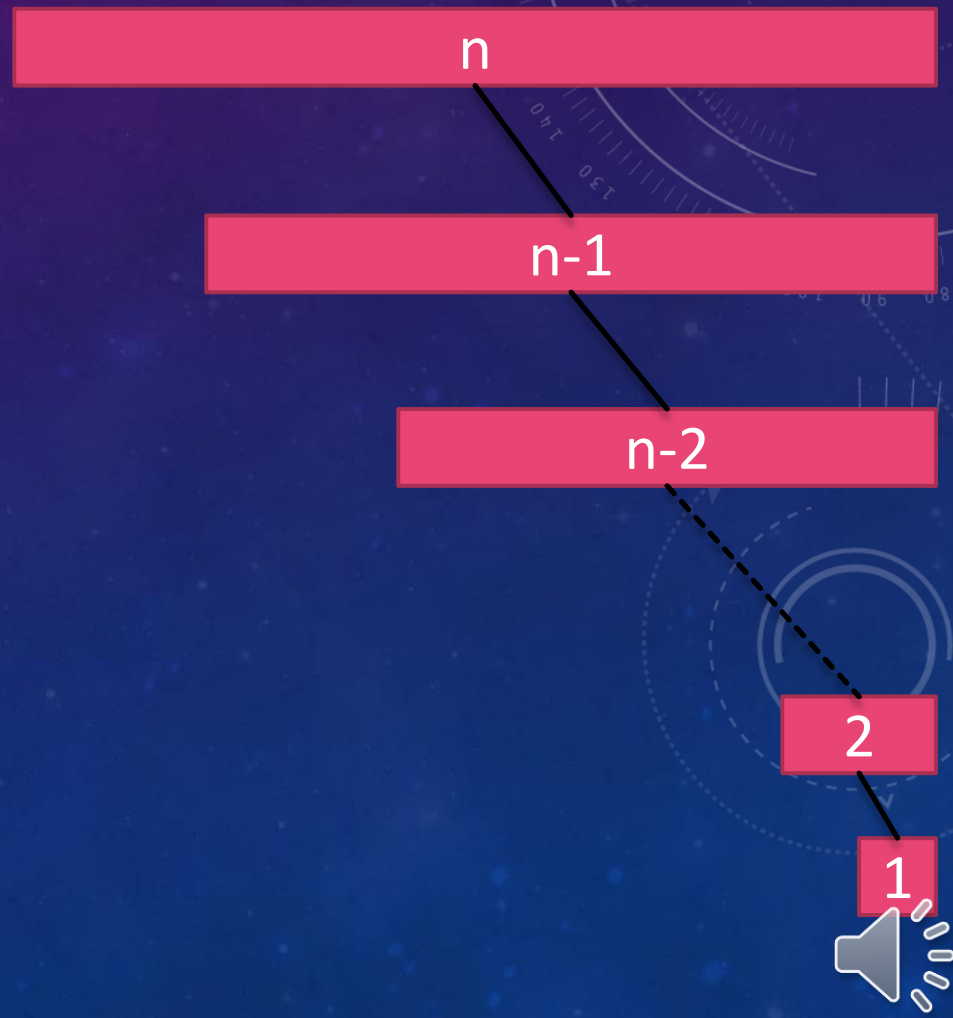
$$T(n) = n + n \log n$$

- The best case of quicksort has a time complexity of $O(n \log n)$.



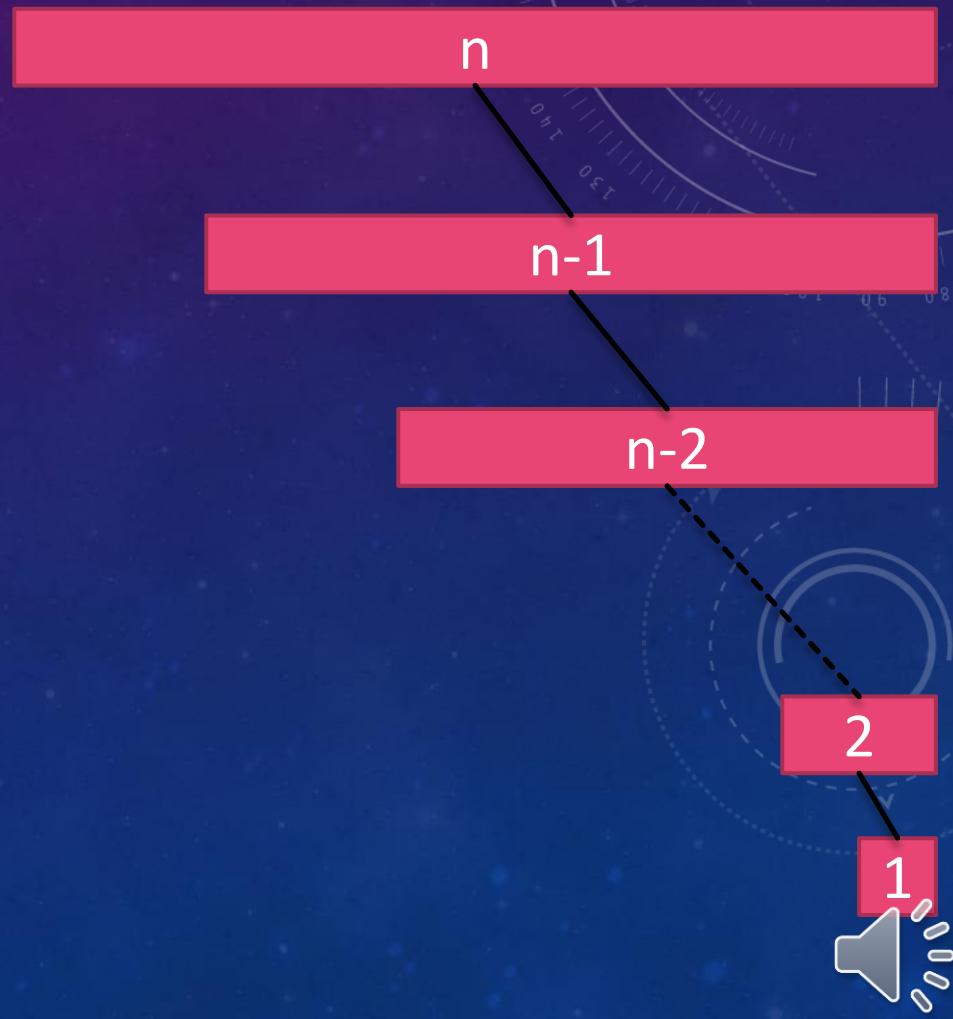
WORST CASE RUNNING TIME OF QUICKSORT

- The worst case of running time of quicksort happens when the sequence is already sorted in ascending or descending order. The correct position of the pivot is at the first or the last index of the sequence in every pass.
- The longest possible subsequence breaks off every time.
- Let the sequence be sorted in ascending order.
- A subsequence of $n - 1$ length breaks off on the right of the pivot in the first pass.
- A subsequence of $n - 2$ length breaks off on the right of the pivot in the second pass.



WORST CASE RUNNING TIME OF QUICKSORT

- Let $T(n)$ be the time taken to sort n elements.
- Then the time taken to sort $n-1$ elements must be $T(n-1)$.
- We compare the first pivot with n elements in the sequence to place it in its correct position, which is the 0^{th} index of the sequence.
- Time to sort one element is constant, so we say $T(1) = 1$.
- Time to sort n elements = Number of comparisons to place the pivot in its correct position + time to sort the subsequence of values greater than the pivot.



WORST CASE RUNNING TIME OF QUICKSORT

- Time to sort n elements = Number of comparisons to place the pivot in its correct position + time to sort the subsequence of values greater than the pivot.

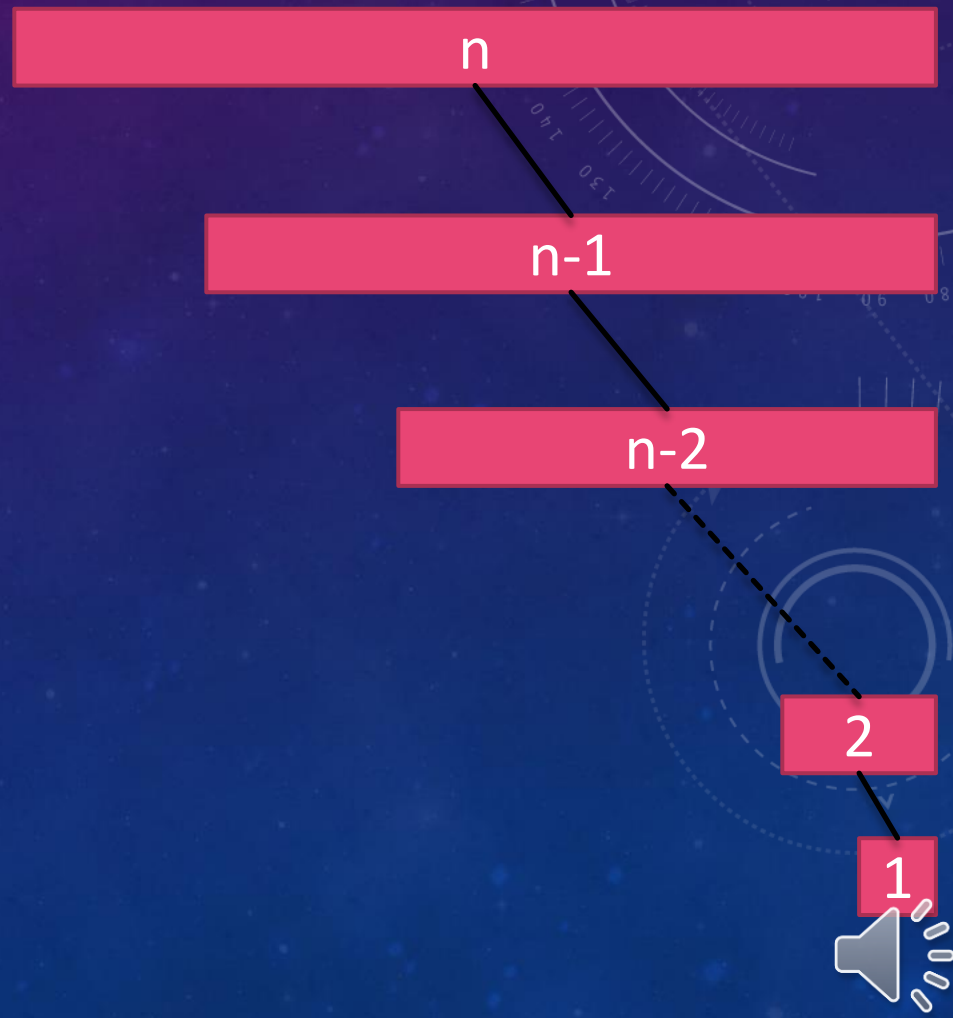
$$T(n) = n + T(n-1), T(1) = 1$$

$$T(n) = T(n-1) + n, T(1) = 1$$

- We will now solve this recurrence relation using backtracking.

$$T(n) = T(n-1) + n$$

$$T(n) = [T(n-2) + (n-1)] + n$$
$$= T(n-2) + (n-1) + n$$



WORST CASE RUNNING TIME OF QUICKSORT

$$T(n) = T(n-1) + n$$

$$T(n) = [T(n-2) + (n-1)] + n$$
$$= T(n-2) + (n-1) + n$$

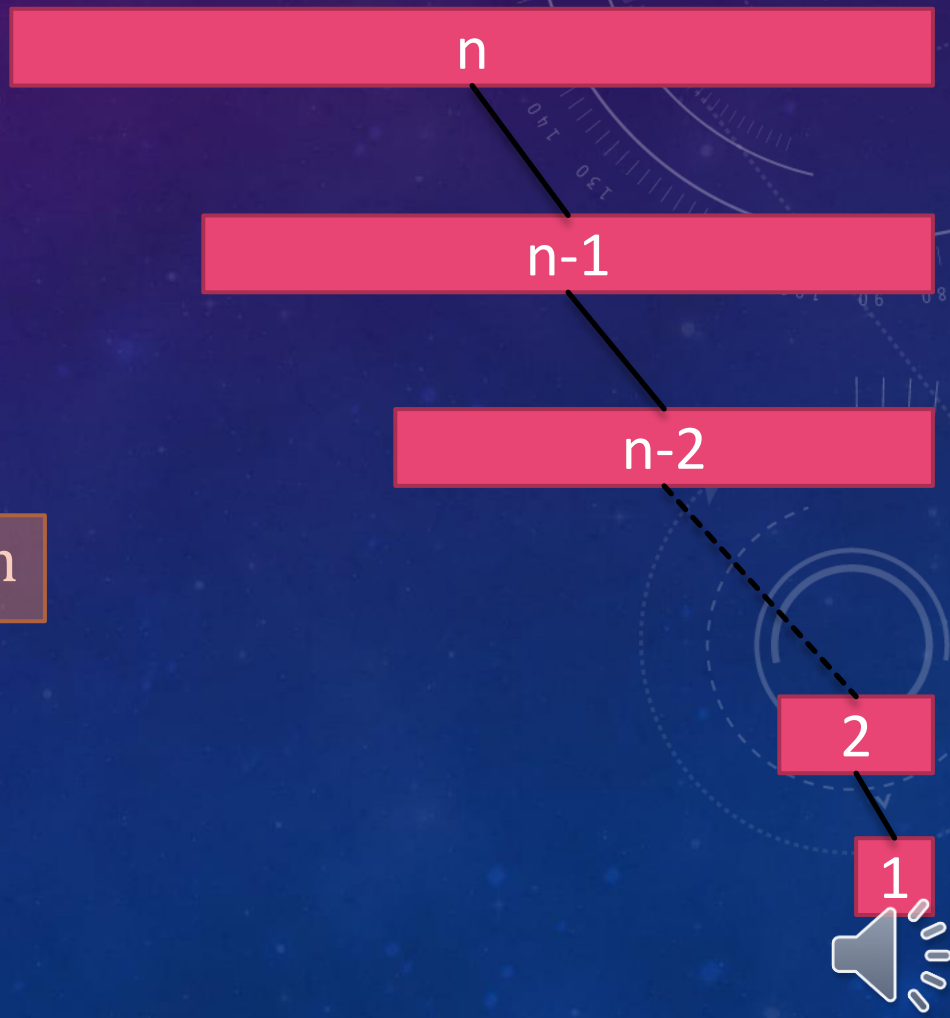
$$T(n) = [T(n-3) + (n-2)] + (n-1) + n$$
$$= T(n-3) + (n-2) + (n-1) + n$$

- Generalizing for k, we can say that:

$$T(n) = T(n-k) + (n-(k-1)) + \cdots + (n-1) + n$$

- There are k terms in this series from $n-(k-1)$ to $n-(0)$. The sum of this series is:

$$S_k = \frac{k}{2}(n-(k-1)+n) = \frac{k}{2}(2n-k+1)$$



WORST CASE RUNNING TIME OF QUICKSORT

$$T(n) = T(n-k) + \frac{k}{2}(2n-k+1)$$

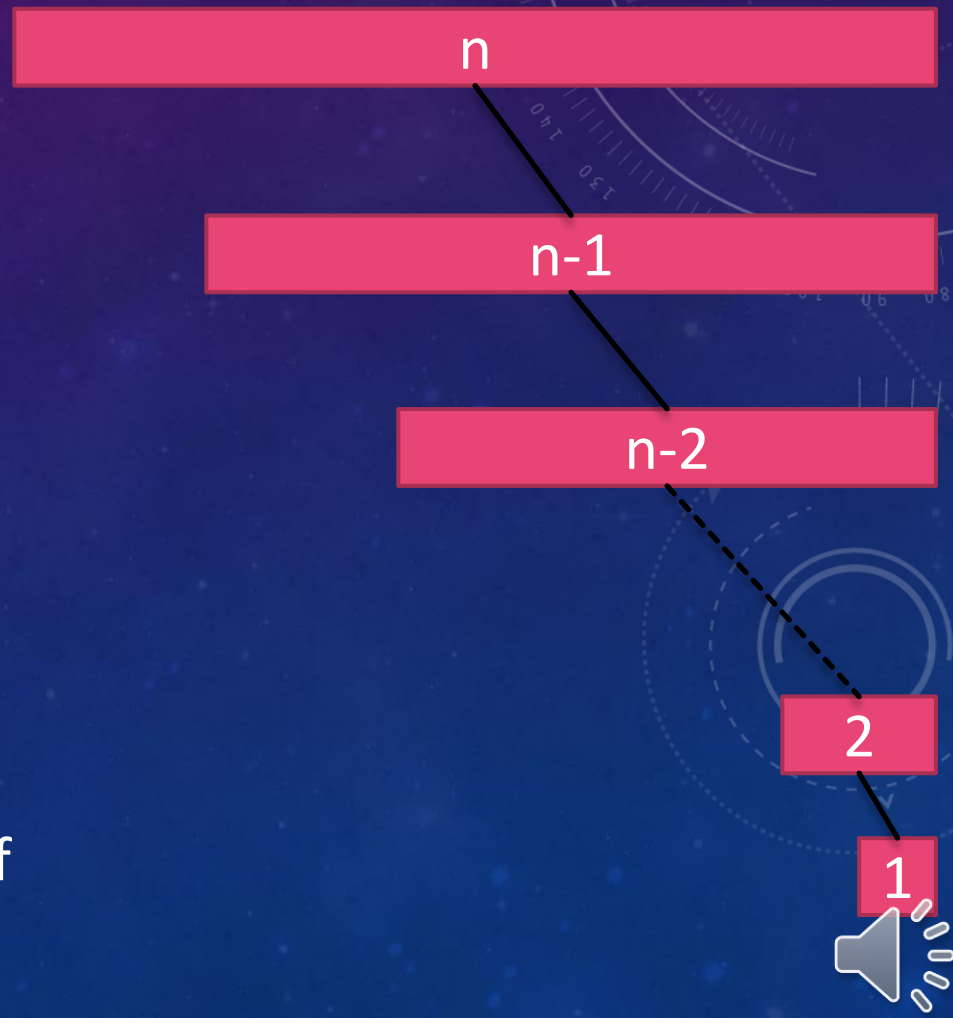
- We want $n-k=1 \Rightarrow k=n-1$.
- Putting $k=n-1$:

$$T(n) = T(n-(n-1)) + \frac{n-1}{2}(2n-(n-1)+1)$$

$$T(n) = T(1) + \frac{n-1}{2}(n+2)$$

$$T(n) = 1 + \frac{n-1}{2}(n+2)$$

- The worst case of quicksort has a time complexity of $O(n^2)$.



SO WHAT DID WE LEARN TODAY?

We were introduced to the quicksort algorithm.

We traced through the quicksort algorithm.

We investigated the time complexity of the algorithm in best and worst cases.

THINGS TO DO

Read the book!

Note your questions and put them up in the relevant online session.

Email suggestions on content or quality of this lecture at uroojain@neduet.edu.pk