Data Structures

Analysis of Quick Sort: Worst case

- When does we encounter the worst case?
  - Input is sorted (ex: 11, 14, 25, 36, 38, 44, 48)
  - Input is reverse sorted (ex: 48, 44, 38, 36, 25, 14, 11)

- What about the performance of insertion in the above mentioned cases

Analysis of insertion sort

for j ← 1 to n-1 do	n
key ← A[j]	n-1
{insert A[j] into the sorted sequence A[0. .j-1]}	
i ← j-1	n-1
while i ≥ 0 and A[i] > key do	$\sum_{j=1}^{n-1} t_j$
A[i+1] ← A[i]	$\sum_{j=1}^{n-1} (t_j - 1)$
i - -	$\sum_{j=1}^{n-1} (tj - 1)$
A[i+1] ← key	n-1

Analysis of insertion sort

T(n) = n + (n-1) + (n-1) + $\sum_{j=1}^{n-1} t_j$+ $\sum_{j=1}^{n-1}(t_j - 1)$+ $\sum_{j=1}^{n-1}(t_j - 1)$+ (n-1)

= n +3(n-1) + (n-1)n/2 + (n-1)(n-2)

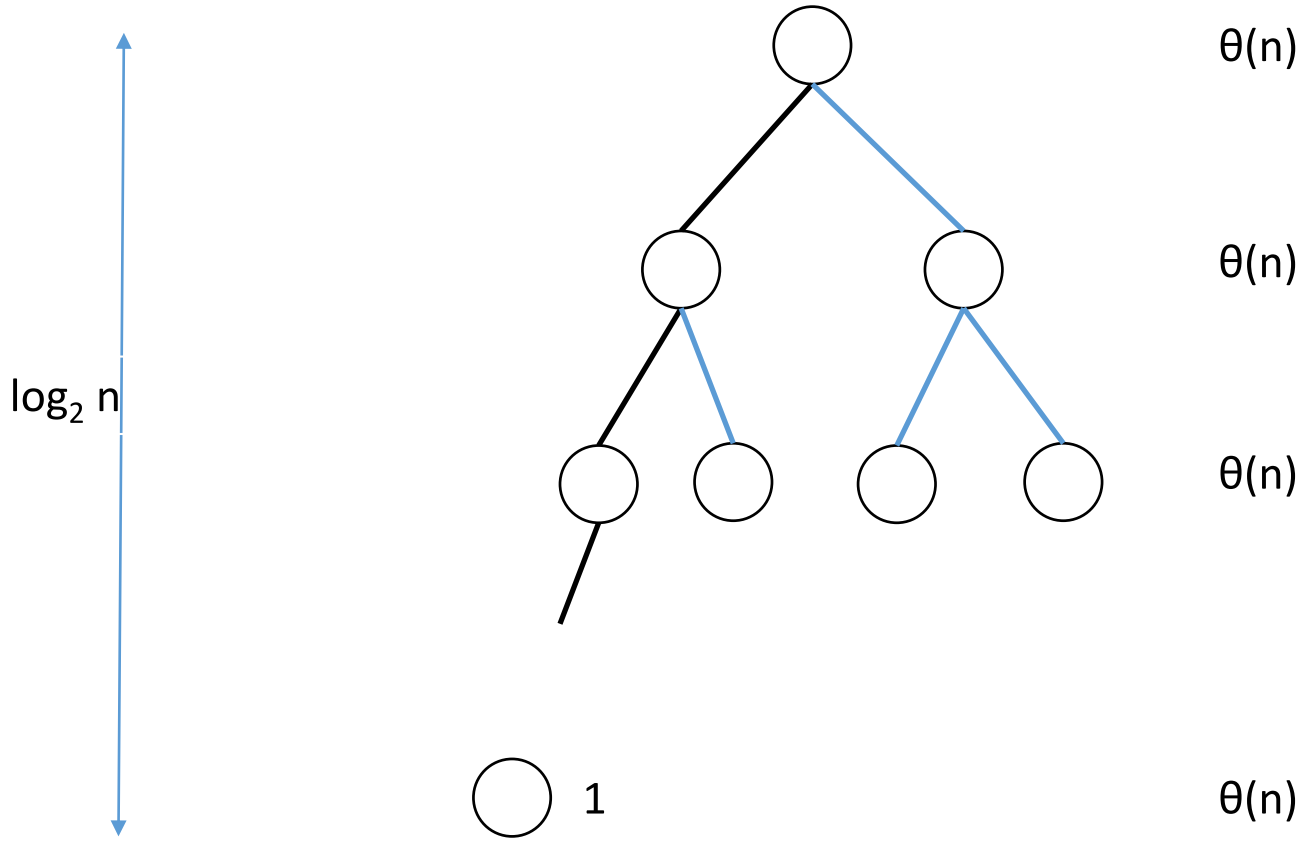
= an^2 + bn + c

Analysis of Quick Sort: Best case

- Consider a sequence with n elements
- The partition step splits the given sequence evenly
- $\lfloor n/2 \rfloor, \lceil n/2 \rceil - 1$

  T(n) = 2T(n/2) + θ(n)
- The height of the recursive tree of quick sort is: θ(log n)

Analysis of Quick Sort: Best case



Analysis of Quick Sort: Best case

- Guess is: $T(n)$ is $\theta(n \log n)$, that is, $T(n) \le c_1 n \log n$ and $T(n) \ge c_2 n \log n$
- Consider the case $T(n) \le c_1 n \log n$

$$\begin{aligned} T(n) &\le 2c_1 (n/2) \log (n/2) + \theta(n) \\ &\le c_1 n \log (n/2) + \theta(n) \\ &\le c_1 n \log n - c_1 n + \theta(n) \\ &\le c_1 n \log n \end{aligned}$$

Analysis of Quick Sort: Best case

- Consider the case $T(n) \geq c_2 n \log n$

$T(n) \geq 2c_2\ (n/2) \log (n/2) + \theta(n)$

$\geq c_2\ n \log (n/2) + \theta(n)$

$\geq c_2\ n \log n - c_2 n + \theta(n)$

$\geq c_2\ n \log n$

Pivot selection

- Popular choice: either the last element or the first element
  - Creates problem when input is in sorted or reverse sorted order
- Safe option: choose the pivot randomly (expensive)
- Median: Median of the array (and median of the first, middle and the last element in the given sequence)

Quick sort: Analysis

```
Algorithm Alt(A, p, r)
    x ← A[r]
    i ← p-1
    for j ← p to r-1
        if(A[j] ≤ x)
            i ← i +1
            exchange A[i] and A[j]
    exchange A[i+1] and A[r]
    return i+1
```

Partitioning

```
    +----------------------------------------+
    | 8   1   4   9   0   3   5   2   7   6  |
    +----------------------------------------+
i     j

    +----------------------------------------+
    | 1   8   4   9   0   3   5   2   7   6  |
    +----------------------------------------+
      i   j

    +----------------------------------------+
    | 1   4   8   9   0   3   5   2   7   6  |
    +----------------------------------------+
          i   j

    +----------------------------------------+
    | 1   4   0   9   8   3   5   2   7   6  |
    +----------------------------------------+
              i       j

    +----------------------------------------+
    | 1   4   0   3   8   9   5   2   7   6  |
    +----------------------------------------+
                  i       j
```

```
    +----------------------------------------+
    | 8   1   4   9   0   3   5   2   7   6  |
    +----------------------------------------+
i         j

    +----------------------------------------+
    | 1   8   4   9   0   3   5   2   7   6  |
    +----------------------------------------+
      i       j

    +----------------------------------------+
    | 1   4   8   9   0   3   5   2   7   6  |
    +----------------------------------------+
          i           j

    +----------------------------------------+
    | 1   4   0   9   8   3   5   2   7   6  |
    +----------------------------------------+
              i           j

    +----------------------------------------+
    | 1   4   0   3   8   9   5   2   7   6  |
    +----------------------------------------+
                  i           j
```

Partitioning

1	4	0	3	5	9	8	2	7	6
				i		j			

1	4	0	3	5	2	8	9	7	6
					i		j		

1	4	0	3	5	2	6	9	7	8
						i		j	

1	4	0	3	5	9	8	2	7	6
				i			j		

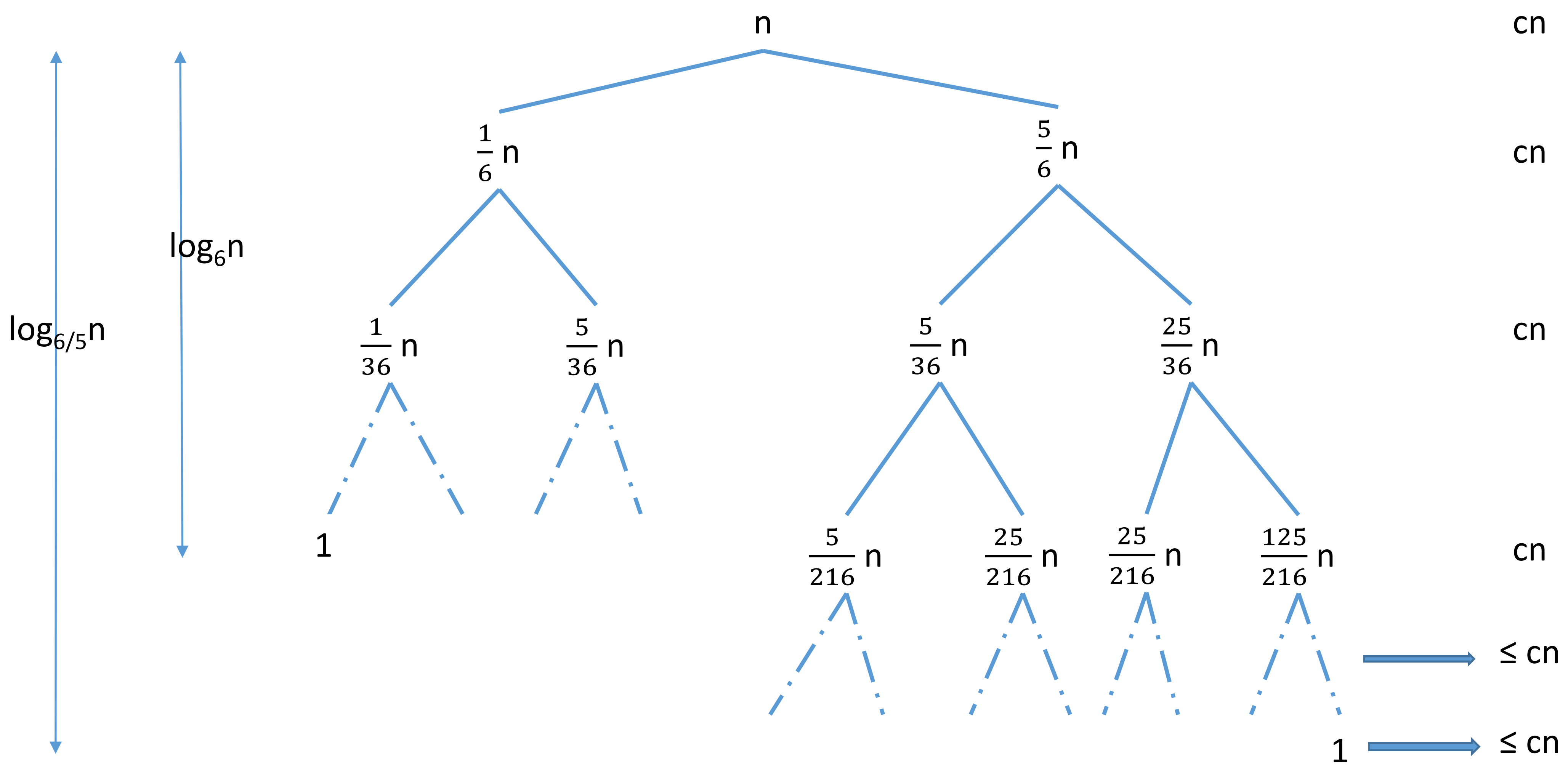
1	4	0	3	5	2	8	9	7	6
					i			j	

Partitioning Algorithm

```
Partition(A, p, r)
        x ← A[r]
        i ← p-1
        j ← r+1
        for(;;)
                while(A[++i] ≤ x) { }
                while(A[--j] ≥ x) { }
                if(i < j)
                        swap(A[i], A[j])
                else
                        break
        exchange(A[i], A[r])
        return i
```

Analysis of Quick Sort

- Assume that the partitioning algorithm produces 5-to-1 proportional split: T(n) = T(n/6) + T(5n/6) + cn



Analysis of Quick Sort

- Assume that the partitioning algorithm produces 9-to-1 proportional split: T(n) = T(n/10) + T(9n/10) + cn