Runtime of Quicksort

COMS10017 - (Object-Oriented Programming and) Algorithms

Dr Christian Konrad

Quicksort

Require: array A of length n
  if $n \leq 1$ **then**
    return A
  else
    $i \leftarrow$ Partition(A)
    $\textsc{Quicksort}(A[0, i-1])$
    $\textsc{Quicksort}(A[i+1, n-1])$

Algorithm $\textsc{Quicksort}$

Require: array A of length n
  if $n \leq 1$ **then**
    return A
  else
    $i \leftarrow$ Partition(A)
    $\textsc{Quicksort}(A[0, i-1])$
    $\textsc{Quicksort}(A[i+1, n-1])$

Algorithm $\textsc{Quicksort}$

Partition A around a Pivot:

14	3	9	8	16	2	1	**7**	11	12	5

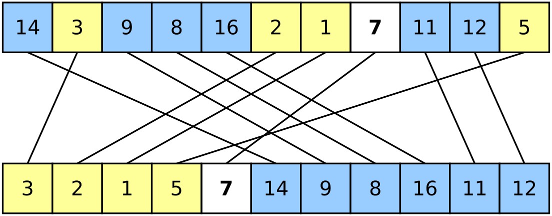
				7						

Quicksort

Require: array A of length n
  if $n \leq 1$ **then**
    return A
  else
    $i \leftarrow \text{Partition}(A)$
    $\textsc{Quicksort}(A[0, i-1])$
    $\textsc{Quicksort}(A[i+1, n-1])$

Algorithm $\textsc{Quicksort}$

Partition A around a Pivot:



Require: array A of length n
  if $n \leq 1$ **then**
    return A
  else
    $i \leftarrow \text{Partition}(A)$
    $\textsc{Quicksort}(A[0, i-1])$
    $\textsc{Quicksort}(A[i+1, n-1])$

Algorithm $\textsc{Quicksort}$

Partition A around a Pivot:

14	3	9	8	16	2	1	**7**	11	12	5

1	2	3	5	**7**	8	9	11	12	14	16

Runtime of Quicksort

Runtime:

Runtime of Quicksort

Runtime: $T(n)$: worst-case runtime on input of length n

Runtime of Quicksort

Runtime: $T(n)$: worst-case runtime on input of length n

$$T(1) = O(1) \qquad \text{(termination condition)}$$

Runtime of Quicksort

Runtime: $T(n)$: worst-case runtime on input of length n

$$\begin{aligned} T(1) &= O(1) \qquad \text{(termination condition)} \\ T(n) &= O(n) + T(n_1) + T(n_2)\ , \end{aligned}$$

where n_1, n_2 are the lengths of the two resulting subproblems.

Runtime of Quicksort

Runtime: $T(n)$: worst-case runtime on input of length n

$$\begin{aligned} T(1) &= O(1) \qquad \text{(termination condition)} \\ T(n) &= O(n) + T(n_1) + T(n_2) \ , \end{aligned}$$

where n_1, n_2 are the lengths of the two resulting subproblems.

Observe: $n_1 + n_2 = n - 1$

Runtime of Quicksort

Runtime: $T(n)$: worst-case runtime on input of length n

$$\begin{aligned} T(1) &= O(1) \qquad \text{(termination condition)} \\ T(n) &= O(n) + T(n_1) + T(n_2) \ , \end{aligned}$$

where n_1, n_2 are the lengths of the two resulting subproblems.

Observe: $n_1 + n_2 = n - 1$

Worst-case:

Runtime of Quicksort

Runtime: $T(n)$: worst-case runtime on input of length n

$$\begin{aligned} T(1) &= O(1) \qquad \text{(termination condition)} \\ T(n) &= O(n) + T(n_1) + T(n_2)\ , \end{aligned}$$

where n_1, n_2 are the lengths of the two resulting subproblems.

Observe: $n_1 + n_2 = n - 1$

Worst-case:

- Suppose that pivot is always the largest element

Runtime of Quicksort

Runtime: $T(n)$: worst-case runtime on input of length n

$$\begin{aligned} T(1) &= O(1) \qquad \text{(termination condition)} \\ T(n) &= O(n) + T(n_1) + T(n_2) \ , \end{aligned}$$

where n_1, n_2 are the lengths of the two resulting subproblems.

Observe: $n_1 + n_2 = n - 1$

Worst-case:

- Suppose that pivot is always the largest element
- Then, $n_1 = n - 1$, $n_2 = 0$

Runtime of Quicksort

Runtime: $T(n)$: worst-case runtime on input of length n

$$\begin{aligned} T(1) &= O(1) \qquad \text{(termination condition)} \\ T(n) &= O(n) + T(n_1) + T(n_2) \ , \end{aligned}$$

where n_1, n_2 are the lengths of the two resulting subproblems.

Observe: $n_1 + n_2 = n - 1$

Worst-case:

- Suppose that pivot is always the largest element
- Then, $n_1 = n - 1$, $n_2 = 0$

Best-case:

Runtime of Quicksort

Runtime: $T(n)$: worst-case runtime on input of length n

$$\begin{aligned} T(1) &= O(1) \qquad \text{(termination condition)} \\ T(n) &= O(n) + T(n_1) + T(n_2)\ , \end{aligned}$$

where n_1, n_2 are the lengths of the two resulting subproblems.

Observe: $n_1 + n_2 = n - 1$

Worst-case:

- Suppose that pivot is always the largest element
- Then, $n_1 = n - 1$, $n_2 = 0$

Best-case:

- Suppose pivot splits array evenly, i.e., pivot is the median

Runtime of Quicksort

Runtime: $T(n)$: worst-case runtime on input of length n

$$\begin{aligned} T(1) &= O(1) \qquad \text{(termination condition)} \\ T(n) &= O(n) + T(n_1) + T(n_2) \ , \end{aligned}$$

where n_1, n_2 are the lengths of the two resulting subproblems.

Observe: $n_1 + n_2 = n - 1$

Worst-case:

- Suppose that pivot is always the largest element
- Then, $n_1 = n - 1$, $n_2 = 0$

Best-case:

- Suppose pivot splits array evenly, i.e., pivot is the median
- Then, $n_1 = \lfloor \frac{n-1}{2} \rfloor$, $n_2 = \lceil \frac{n-1}{2} \rceil$

Quicksort: Worst case

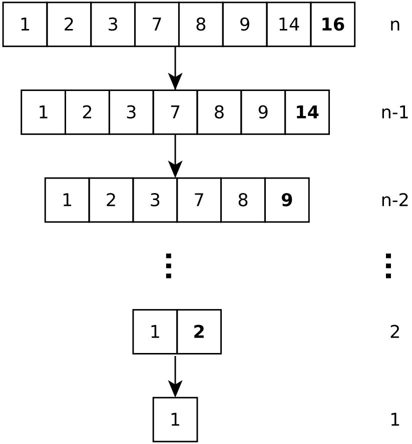
Partition:

Quicksort: Worst case

Partition: Let C be such that Partition() runs in time at most Cn

Quicksort: Worst case

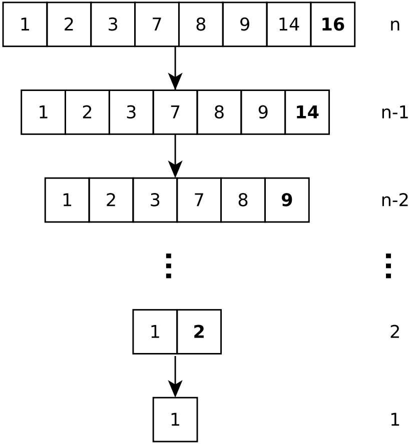
Partition: Let C be such that Partition() runs in time at most Cn



Quicksort: Worst case

Partition: Let C be such that Partition() runs in time at most Cn

Recurrence:

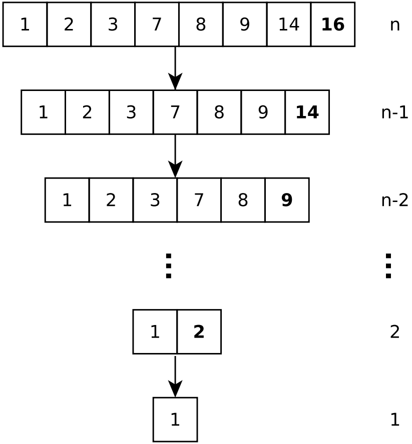


Quicksort: Worst case

Partition: Let C be such that Partition() runs in time at most Cn

Recurrence:

$$T(n) \leq Cn + T(n-1)$$



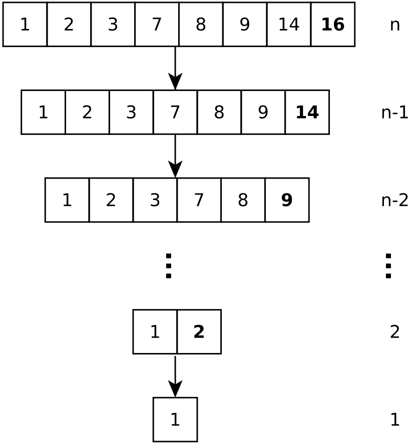
Quicksort: Worst case

Partition: Let C be such that Partition() runs in time at most Cn

Recurrence:

$$T(n) \leq Cn + T(n-1)$$

Total Runtime:



Quicksort: Worst case

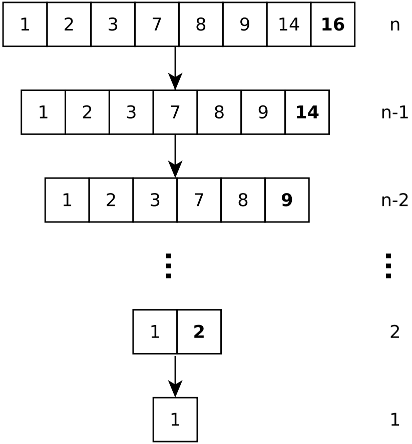
Partition: Let C be such that Partition() runs in time at most Cn

Recurrence:

$$T(n) \leq Cn + T(n-1)$$

Total Runtime:

$T(n)$



Quicksort: Worst case

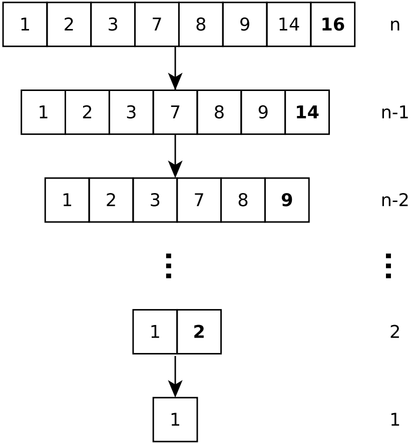
Partition: Let C be such that Partition() runs in time at most Cn

Recurrence:

$$T(n) \leq Cn + T(n-1)$$

Total Runtime:

$$T(n) \leq \sum_{i=1}^{n} Ci = C \sum_{i=1}^{n} i$$



Quicksort: Worst case

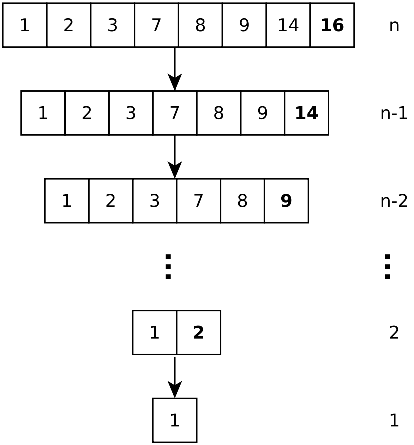
Partition: Let C be such that Partition() runs in time at most Cn

Recurrence:

$$T(n) \leq Cn + T(n-1)$$

Total Runtime:

$$T(n) \leq \sum_{i=1}^{n} Ci = C \sum_{i=1}^{n} i = C\frac{(n+1)n}{2}$$



Quicksort: Worst case

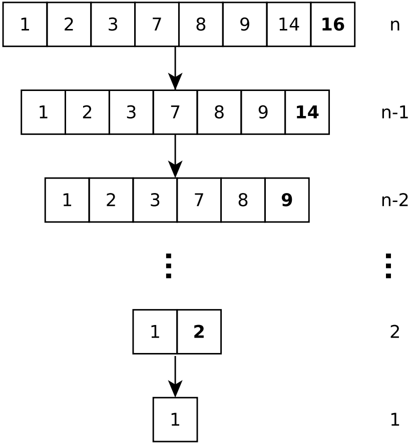
Partition: Let C be such that Partition() runs in time at most Cn

Recurrence:

$$T(n) \leq Cn + T(n-1)$$

Total Runtime:

$$\begin{aligned} T(n) &\leq \sum_{i=1}^{n} Ci = C \sum_{i=1}^{n} i \\ &= C\frac{(n+1)n}{2} \\ &= \frac{C}{2}(n^2+n) \end{aligned}$$



Quicksort: Worst case

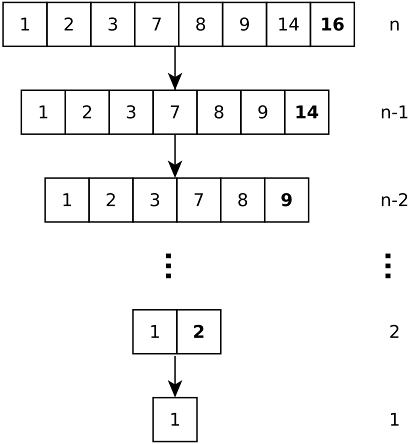
Partition: Let C be such that Partition() runs in time at most Cn

Recurrence:

$$T(n) \leq Cn + T(n-1)$$

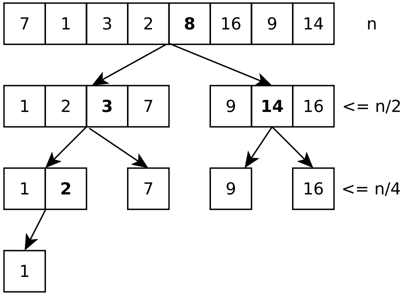
Total Runtime:

$$\begin{aligned} T(n) &\leq \sum_{i=1}^{n} Ci = C \sum_{i=1}^{n} i \\ &= C \frac{(n+1)n}{2} \\ &= \frac{C}{2}(n^2 + n) = O(n^2) \ . \end{aligned}$$



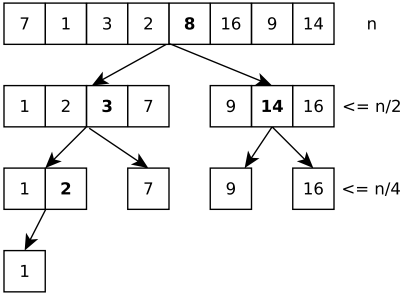
Quicksort: Best case

Best Case:



Quicksort: Best case

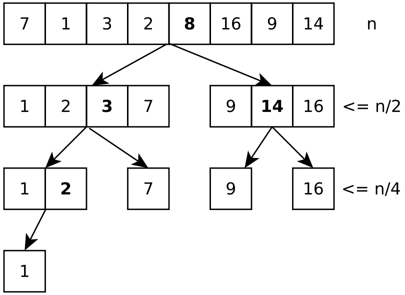
Best Case: $n_1, n_2 \leq \frac{n}{2}$



Quicksort: Best case

Best Case: $n_1, n_2 \leq \frac{n}{2}$

Number of Levels: ℓ

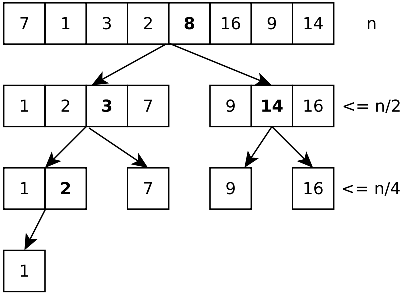


Quicksort: Best case

Best Case: $n_1, n_2 \leq \frac{n}{2}$

Number of Levels: ℓ

- Last level: $n = 1$



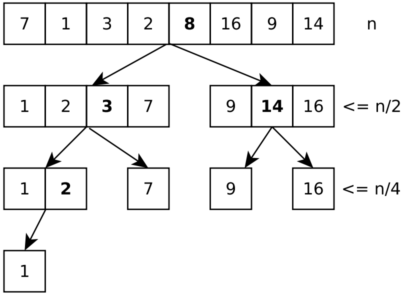
Quicksort: Best case

Best Case: $n_1, n_2 \leq \frac{n}{2}$

Number of Levels: ℓ

- Last level: $n = 1$

$$\frac{n}{2^{\ell-1}} \leq 1$$



Quicksort: Best case

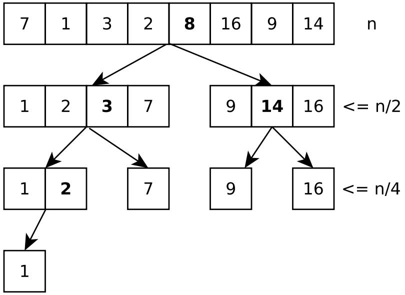
Best Case: $n_1, n_2 \leq \frac{n}{2}$

Number of Levels: ℓ

- Last level: $n = 1$

$$\frac{n}{2^{\ell-1}} \leq 1$$

$$\log(n) + 1 \leq \ell$$



Quicksort: Best case

Best Case: $n_1, n_2 \leq \frac{n}{2}$

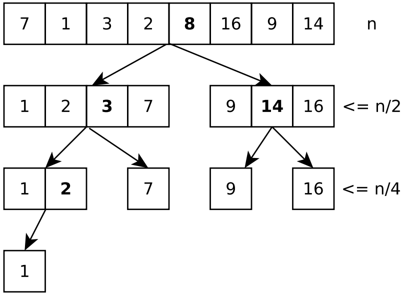
Number of Levels: ℓ

- Last level: $n = 1$

$$\frac{n}{2^{\ell-1}} \leq 1$$

$$\log(n) + 1 \leq \ell$$

- Last but one level: $n = 2$



Quicksort: Best case

Best Case: $n_1, n_2 \leq \frac{n}{2}$

Number of Levels: ℓ

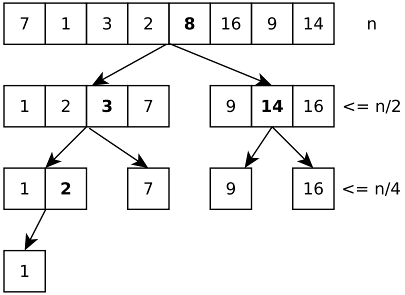
- Last level: $n = 1$

$$\frac{n}{2^{\ell-1}} \leq 1$$

$$\log(n) + 1 \leq \ell$$

- Last but one level: $n = 2$

$$\frac{n}{2^{\ell-2}} > 1$$



Quicksort: Best case

Best Case: $n_1, n_2 \leq \frac{n}{2}$

Number of Levels: ℓ

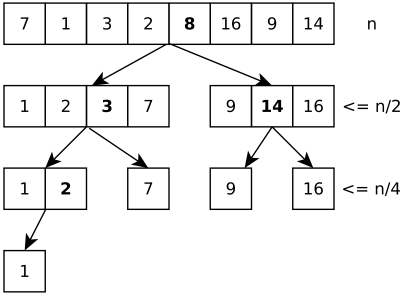
- Last level: $n = 1$

$$\frac{n}{2^{\ell-1}} \leq 1$$

$$\log(n) + 1 \leq \ell$$

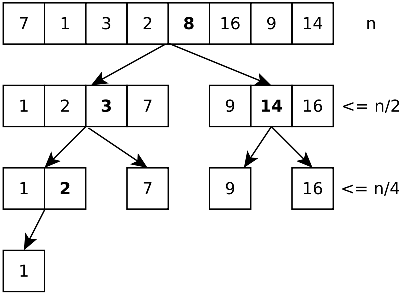
- Last but one level: $n = 2$

$$\frac{n}{2^{\ell-2}} > 1 \text{ which implies } \log(n) + 2 > \ell$$



Quicksort: Best case

Best Case: $n_1, n_2 \leq \frac{n}{2}$



Number of Levels: ℓ

- Last level: $n = 1$

$$\frac{n}{2^{\ell-1}} \leq 1$$

$$\log(n) + 1 \leq \ell$$

- Last but one level: $n = 2$

$$\frac{n}{2^{\ell-2}} > 1 \text{ which implies } \log(n) + 2 > \ell$$

- Hence, there are $\ell = \lceil \log(n) \rceil + 1$ levels

Quicksort: Best case

Best Case: $n_1, n_2 \le \frac{n}{2}$

Number of Levels: ℓ

- Last level: $n = 1$

$$\frac{n}{2^{\ell-1}} \le 1$$

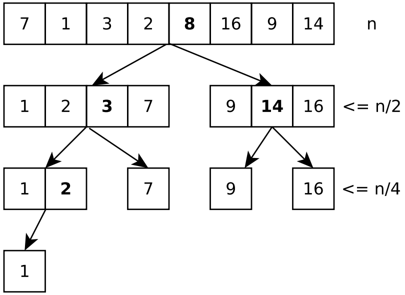
$$\log(n) + 1 \le \ell$$

- Last but one level: $n = 2$

$$\frac{n}{2^{\ell-2}} > 1 \text{ which implies } \log(n) + 2 > \ell$$

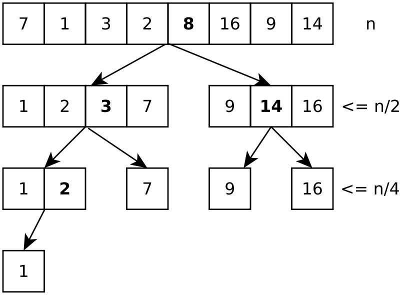
- Hence, there are $\ell = \lceil \log(n) \rceil + 1$ levels

Total Runtime:



Quicksort: Best case

Best Case: $n_1, n_2 \leq \frac{n}{2}$



Number of Levels: ℓ

- Last level: $n = 1$

$$\frac{n}{2^{\ell-1}} \leq 1$$

$$\log(n) + 1 \leq \ell$$

- Last but one level: $n = 2$

$$\frac{n}{2^{\ell-2}} > 1 \text{ which implies } \log(n) + 2 > \ell$$

- Hence, there are $\ell = \lceil \log(n) \rceil + 1$ levels

Total Runtime:

- Observe: Total runtime of Partition() in a level: $O(n)$

Quicksort: Best case

Best Case: $n_1, n_2 \leq \frac{n}{2}$

Number of Levels: ℓ

- Last level: $n = 1$

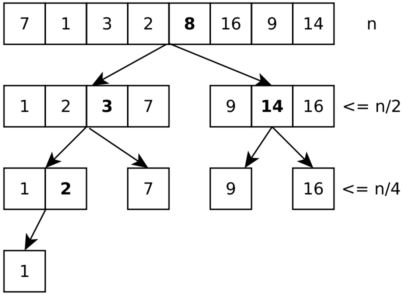
$$\frac{n}{2^{\ell-1}} \leq 1$$

$$\log(n) + 1 \leq \ell$$

- Last but one level: $n = 2$

$$\frac{n}{2^{\ell-2}} > 1 \text{ which implies } \log(n) + 2 > \ell$$

- Hence, there are $\ell = \lceil \log(n) \rceil + 1$ levels



Total Runtime:

- Observe: Total runtime of Partition() in a level: $O(n)$
- Total runtime: $\ell \cdot O(n) = O(n \log n)$.

Runtime: Discussion

Good versus Bad Splits:

Runtime: Discussion

Good versus Bad Splits:

- It is crucial that subproblems are *roughly* balanced

Runtime: Discussion

Good versus Bad Splits:

- It is crucial that subproblems are *roughly* balanced
- In fact, enough if $n_1 = \frac{1}{1000}n$ and $n_2 = n - 1 - n_1$ to get a runtime of $O(n \log n)$

Runtime: Discussion

Good versus Bad Splits:

- It is crucial that subproblems are *roughly* balanced
- In fact, enough if $n_1 = \frac{1}{1000}n$ and $n_2 = n - 1 - n_1$ to get a runtime of $O(n \log n)$
- Even enough if subproblems roughly balanced *most of the time*

Runtime: Discussion

Good versus Bad Splits:

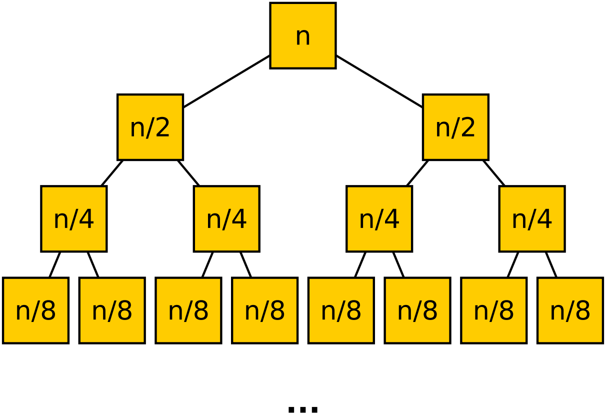
- It is crucial that subproblems are *roughly* balanced
- In fact, enough if $n_1 = \frac{1}{1000} n$ and $n_2 = n - 1 - n_1$ to get a runtime of $O(n \log n)$
- Even enough if subproblems roughly balanced *most of the time*
- In practice, this happens most of the time, $\textsc{Quicksort}$ is therefore usually very fast

Runtime: Discussion

Good versus Bad Splits:

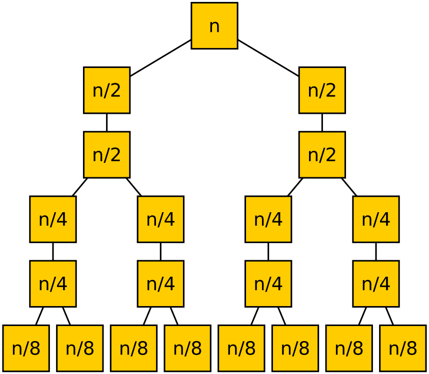
- It is crucial that subproblems are *roughly* balanced
- In fact, enough if $n_1 = \frac{1}{1000} n$ and $n_2 = n - 1 - n_1$ to get a runtime of $O(n \log n)$
- Even enough if subproblems roughly balanced *most of the time*
- In practice, this happens most of the time, $\textsc{Quicksort}$ is therefore usually very fast

Good versus Bad Splits: Intuition and Rough Analysis



Only good splits: Recursion tree depth $\lceil \log n \rceil + 1$

Good versus Bad Splits: Intuition and Rough Analysis



...

Good & bad splits alternate: Recursion tree depth $2 \cdot (\lceil \log n \rceil + 1)$

Selecting good Pivots

Ideal Pivot:

Selecting good Pivots

Ideal Pivot: Median

Selecting good Pivots

Ideal Pivot: Median

Pivot Selection

Selecting good Pivots

Ideal Pivot: Median

Pivot Selection

- To obtain runtime of $O(n \log n)$, we can spend $O(n)$ time to select a good pivot

Selecting good Pivots

Ideal Pivot: Median

Pivot Selection

- To obtain runtime of $O(n \log n)$, we can spend $O(n)$ time to select a good pivot
- There are $O(n)$ time algorithms for finding the median

Selecting good Pivots

Ideal Pivot: Median

Pivot Selection

- To obtain runtime of $O(n \log n)$, we can spend $O(n)$ time to select a good pivot
- There are $O(n)$ time algorithms for finding the median
- They are complicated and not efficient in practice

Selecting good Pivots

Ideal Pivot: Median

Pivot Selection

- To obtain runtime of $O(n \log n)$, we can spend $O(n)$ time to select a good pivot
- There are $O(n)$ time algorithms for finding the median
- They are complicated and not efficient in practice
- However, using such an algorithm gives $O(n \log n)$ worst case runtime!

Selecting good Pivots

Ideal Pivot: Median

Pivot Selection

- To obtain runtime of $O(n \log n)$, we can spend $O(n)$ time to select a good pivot
- There are $O(n)$ time algorithms for finding the median
- They are complicated and not efficient in practice
- However, using such an algorithm gives $O(n \log n)$ worst case runtime!

Idea that works in Practice:

Selecting good Pivots

Ideal Pivot: Median

Pivot Selection

- To obtain runtime of $O(n \log n)$, we can spend $O(n)$ time to select a good pivot
- There are $O(n)$ time algorithms for finding the median
- They are complicated and not efficient in practice
- However, using such an algorithm gives $O(n \log n)$ worst case runtime!

Idea that works in Practice: Select Pivot at random!

Selecting good Pivots

Ideal Pivot: Median

Pivot Selection

- To obtain runtime of $O(n \log n)$, we can spend $O(n)$ time to select a good pivot
- There are $O(n)$ time algorithms for finding the median
- They are complicated and not efficient in practice
- However, using such an algorithm gives $O(n \log n)$ worst case runtime!

Idea that works in Practice: Select Pivot at random!
(Implementation: exchange $A[n-1]$ with a uniform random element $A[i]$)

Random Pivot Selection

Randomized Algorithm

Random Pivot Selection

Randomized Algorithm

- Randomized pivot selection turns Quicksort into a *Randomized Algorithm*

Random Pivot Selection

Randomized Algorithm

- Randomized pivot selection turns Quicksort into a *Randomized Algorithm*
- Worst-case runtime:

Random Pivot Selection

Randomized Algorithm

- Randomized pivot selection turns Quicksort into a *Randomized Algorithm*
- Worst-case runtime: still $O(n^2)$ (we may be unlucky!)

Random Pivot Selection

Randomized Algorithm

- Randomized pivot selection turns Quicksort into a *Randomized Algorithm*
- Worst-case runtime: still $O(n^2)$ (we may be unlucky!)
- *Expected runtime*: Since we introduce randomness, the runtime of the algorithm becomes a random variable

Random Pivot Selection

Randomized Algorithm

- Randomized pivot selection turns Quicksort into a *Randomized Algorithm*
- Worst-case runtime: still $O(n^2)$ (we may be unlucky!)
- *Expected runtime*: Since we introduce randomness, the runtime of the algorithm becomes a random variable

Definition (Bad Split)

Random Pivot Selection

Randomized Algorithm

- Randomized pivot selection turns Quicksort into a *Randomized Algorithm*
- Worst-case runtime: still $O(n^2)$ (we may be unlucky!)
- *Expected runtime*: Since we introduce randomness, the runtime of the algorithm becomes a random variable

Definition (Bad Split)
A split is *bad* if $\min\{n_1, n_2\} \leq \frac{1}{10} n$.

Random Pivot Selection

Randomized Algorithm

- Randomized pivot selection turns Quicksort into a *Randomized Algorithm*
- Worst-case runtime: still $O(n^2)$ (we may be unlucky!)
- *Expected runtime*: Since we introduce randomness, the runtime of the algorithm becomes a random variable

Definition (Bad Split)
A split is *bad* if $\min\{n_1, n_2\} \leq \frac{1}{10} n$.

If we select the pivot randomly, how likely is it to have a bad split?

Probability of a Bad Split

Probability of a Bad Split

Probability of a Bad Split

- Bad split if element chosen as pivot is either among smallest 0.1 fraction of elements or among largest 0.1 fraction
- Since our choice is random, this happens with probability 0.2

Probability of a Bad Split

Probability of a Bad Split

- Bad split if element chosen as pivot is either among smallest 0.1 fraction of elements or among largest 0.1 fraction
- Since our choice is random, this happens with probability 0.2
- Hence, in average only 1 out of 5 splits are bad

Probability of a Bad Split

Probability of a Bad Split

- Bad split if element chosen as pivot is either among smallest 0.1 fraction of elements or among largest 0.1 fraction
- Since our choice is random, this happens with probability 0.2
- Hence, in average only 1 out of 5 splits are bad
- Hence, 4 out of 5 times the algorithm makes enough *progress*

Probability of a Bad Split

- Bad split if element chosen as pivot is either among smallest 0.1 fraction of elements or among largest 0.1 fraction
- Since our choice is random, this happens with probability 0.2
- Hence, in average only 1 out of 5 splits are bad
- Hence, 4 out of 5 times the algorithm makes enough *progress*

Random Pivot Selection: QUICKSORT runs in expected time $O(n \log n)$ if the pivot is chosen uniformly at random