6: QuickSort (divide and conquer/randomized)

Friday, March 19, 2021 2:47 PM

QuickSort

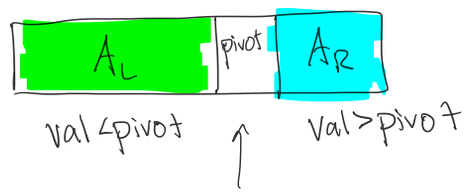
Input: Array A of unique integers

Output: sorted A.

"Randomly chosen"

- If $|A| = 1$: Return A ← Base Case
- pivot = Randomly chosen elt } Preprocessing
- Partition (A, pivot)
- QuickSort (A_L) } Divide + Conquer
- QuickSort (A_R)

Effect of Partition:

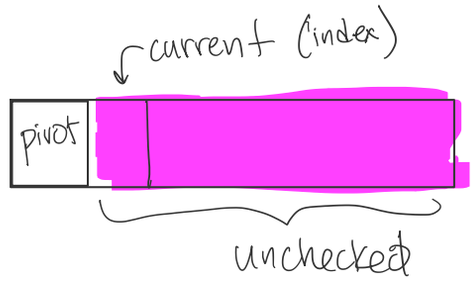


Pivot not included in recursive calls... it is in the correct position

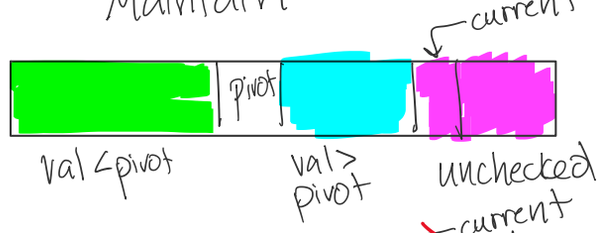
Partition (A, pivot)

value of pivot

- Move pivot to start (swap first element with pivot)
- While current ≤ |A|

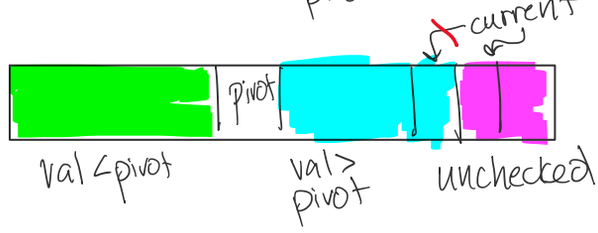


Maintain

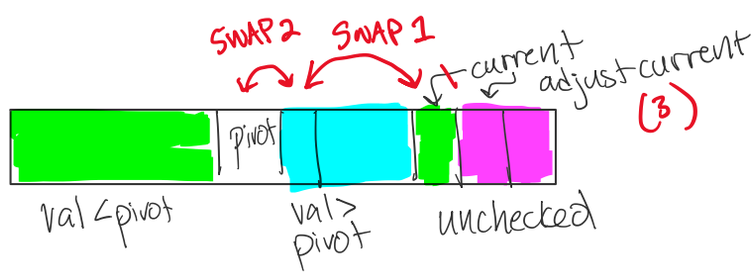


COMPARISON

$\boxed{\text{If } A[\text{current}] > \text{pivot}}$



Else (A[current] < pivot)



Key Pts

- Partition does most of work in QuickSort
- Runtime of Partition $\approx$ # of comparisons

Idea: To determine QuickSort runtime, we will count # of comparisons

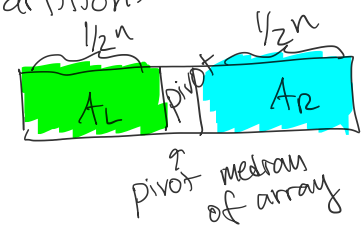
Q: How many comparisons are done by partition on an array of size n?

A) $O(\sqrt{n})$ B) $O(n)$ C) $O(n \log n)$ D) $O(n^2)$

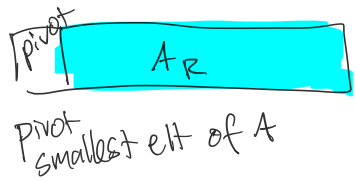
pivot is compared to each elt in array : $(n-1)$ comparisons

Effect of Partition:

Lucky:



Unlucky:



1. Suppose you got very lucky and pivot is always chosen to be median of A, <u>every time</u> partition is called.
   - Create recurrence relation for runtime of Quicksort
   - Solve

2. Suppose you got very unlucky and pivot is always chosen to be minimum of A, <u>every time</u> partition is called.
   - Create recurrence relation for runtime of QuickSort
   - Solve

1. $T(n) = O(n) + 2T(n/2)$ $\qquad T(1) = O(1)$

$n = |A|$ $\qquad$ Partition $\qquad$ A_L, A_R each have $n/2$ elements

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

2. $T(n) = O(n) + T(n-1) + \cancel{T(0)}$ $\leftarrow$ A_L has 0 elements

Partition $\qquad$ A_R has $n-1$ elements

$$= O(n) + O(n-1) + T(n-2)$$

$$= O(n) + O(n-1) + O(n-2) + T(n-3)$$

$$\vdots$$

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

$$\Rightarrow \sum_{i=1}^{n} i = O(n^2)$$

Lucky	Unlucky
$O(n \log n)$	$O(n^2)$

? Which is likely?

? Which is <u>average</u>?

Q's

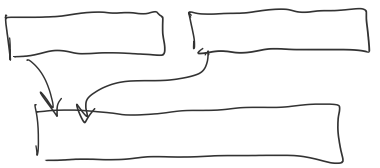
- Why not choose pivot to be median, instead of random?
  - $n/2^{nd}$ largest elt
  - ↳ non-randomized algs. to find median take $O(n \log n)$ time.
- Why not just pick pivot to be middle instead of random
  - index $= n/2$

↳ If your input array has structure, might produce worst case, unlucky behavior

- Complexity of choosing random elt of list? $O(\log n)$
  ↳ need $\log n$ bits to specify location, generate 1 random bit takes $O(1)$ time.

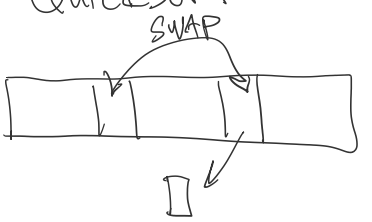
- Space-time tradeoff:

Merge Sort



At each recursive call, creating new array to store merge. Uses a lot of memory, or need to keep reallocating memory, complex process, time consuming

QuickSort



$O(1)$ extra space to store elt value while swapping

- Data Structure Tradeoff:

Mergesort does more moves, fewer comparisons than QuickSort
↳ In a data structure with pointers (linked list), moving elts might be faster than comparing, so MergeSort might be better.

https://stackoverflow.com/questions/47878256/quicksort-and-mergesort-performance-on-sequential-data-fit-in-memory-vs-slow-to

- InsertionSort faster than MergeSort/QuickSort for $< 7, 10$ elts. Switch to Insertion for base case @ 7,10 size for faster runtime. (Betrayed! why learn $O(n^2)$ sort? Bubble → Greedy, Insertion → Small Array, Selection: Hand

- FYI: there is a technique called <u>median of medians</u> to help you choose pivot as elt close enough to median to guarantee $O(n \log n)$ runtime. (Might have time?)

- We will show <u>avg.</u> runtime is $O(n \log n)$

Calculating Average Runtime

1. Describe sample space S.

   Set of all possible sequences of random outcomes that might occur over course of alg. (including in all recursive calls)

2. Create random variable $R: S \to \mathbb{Z}^+$ to track runtime

   function $R(\sigma)$ element of S

3. Evaluate $\mathbb{E}[R]$ (too hard to do directly)

   - Write $R(\sigma) = \sum_i X_i(\sigma)$ ← simpler random variables
   - Use linearity of expectation:

$$\mathbb{E}[R] = \sum_i \mathbb{E}[X_i]$$

1. evaluate $\mathbb{E}[X_i]$

2. evaluate sum

Calculating Average Runtime

1. Describe sample space S.

   Set of all possible sequences of $\boxed{\text{random outcomes}}$ that might occur over course of alg. (including in all recursive calls)

pivot choices

What is the sample space if QuickSort is run on the array:

8	5	7

S = set of possible pivot choices of the algorithm

A) $S = \{8, 5, 7\}$

B) $S = \{(8,5,7), (8,7,5), (5,8,7), (5,7,8), (7,5,8), (7,8,5)\}$

$\boxed{C)}$ $S = \{7, \underbrace{(5,7), (5,8)}, \underbrace{(8,5), (8,7)}\}$

If 7 chosen: $\boxed{5|7|8}$ → $\boxed{5}$ $\boxed{8}$ Arrays too small, no partition

If 5 chosen: $\boxed{5|8|7}$ → $\boxed{8|7}$ 8 or 7 can be chosen as pivot

If 8 chosen $\boxed{5|7|8}$ → $\boxed{5|7}$ 5 or 7 can be chosen as pivot

For large arrays... it will be a complicated!

Instead: S = set of all possible sequences of pivot choices made over course of alg.

2. Create random variable $R: S \to \mathbb{Z}^+$ to track runtime

$R(\sigma)$ = # of comparisons over whole alg. if σ is sequence of pivot choices

From above example: $R(7) < R(5,7)$

1 level of recursion — 2 levels of recursion

3. Evaluate $\mathbb{E}[R]$ (too hard to do directly)

$$\sum_{\sigma \in S} \Pr(\sigma) R(\sigma)$$

Just figuring this out is hard!

- Write $R = \sum_i X_i$ ← simpler random variable

- Use linearity of expectation:

$$\mathbb{E}[R] = \sum_i \mathbb{E}[X_i]$$

1. evaluate $\mathbb{E}[X_i]$

then

2. evaluate sum

$X_{ij}(\sigma)$ = # of comparisons between i^{th} largest + j^{th} largest elts of A, over course of alg, when σ is sequence of pivot choices

z_i = i^{th} smallest element

ex: $\boxed{8 \mid 5 \mid 7}$ $\quad X_{23}$ = # of times 7, 8 are compared over course of alg.

$z_3 = 8 \quad z_1 = 5 \quad z_2 = 7$

$$R(\sigma) = \sum_{1 \le i < j \le n} X_{ij}(\sigma)$$

total # comparisons is sum over number of comparisons for each pair of elts

further abbreviate $\sum_{i<j}$

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n}$$

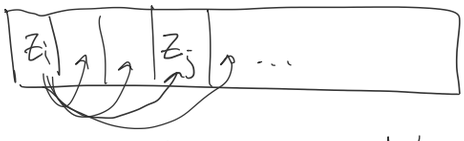
- Suppose z_i, z_j $(i < j)$ are both in an array that is the input to some recursive call of QuickSort. In each of the following cases, when are z_i, z_j compared or not compared in the current recursive call? When are they kept together in the same subarray for the next recursive call or separated?
  - z_i or z_j is chosen as pivot.
  - z_k is chosen as pivot, $k > i, j$
  - z_k is chosen as pivot, $k < i, j$
  - z_k is chosen as pivot, $i < k < j$
- What values can X_{ij} take? (Hint: only two possible values.)

What conditions cause each value?

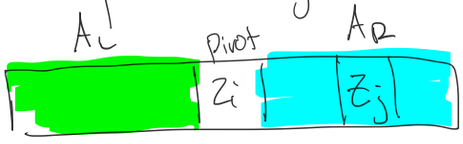
- What is probability of z_i, z_j being compared?

Story of z_i, z_j:

- If z_i or z_j pivot, compared once! COMPARE



... but never compared again b/c

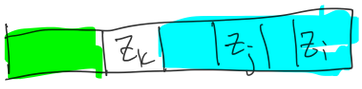


pivot not included in recursive calls

- If pivot $= z_k$, and $k < i, j$, $i, j < k$ Delay

NOT compared.

Every comparison involves pivot

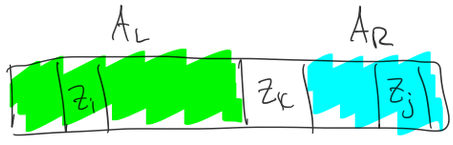


z_i, z_j put together in same subarray for next recursive call... could be compared in future

- If pivot $= z_k$ $i < k < j$ NO COMPARE

NOT compared.

Every comparison involves pivot



Never compared again → in separate arrays for future recursive calls

$X_{ij} = 0$ or 1 *Indicator Random Variable*

Calculating Average Runtime:

$$R = \sum_{i<j} X_{ij}$$

$$\mathbb{E}[R] = \sum_{i<j} \mathbb{E}[X_{ij}]$$

$$= \sum_{i<j} \sum_{\sigma \in S} X_{ij}(\sigma) \cdot \Pr(\sigma)$$

0 or 1 (indicator random variable

$$= \sum_{i<j} \sum_{\substack{\sigma \in S: \\ z_i, z_j \text{ compares}}} \Pr(\sigma)$$

$$= \sum_{i<j} \Pr(z_i, z_j \text{ compared})$$

pivot : compare

$$z_1 \quad z_2 \quad z_3 \quad \cdots \quad \boxed{z_i \quad \underbrace{z_{i+1} \quad \cdots \quad z_{j-1}}_{\text{pivot: no compare}} \quad z_j} \quad \cdots \quad z_n$$

pivot: kicking decision down the road

- there are $j-i+1$ elt in this set
- all equally likely to be chosen, one will always be chosen

What is the probability that z_i, z_j are compared?

A. $\frac{1}{j-i}$ B. $\frac{2}{j-i}$ C. $\frac{1}{j-i+1}$ D. $\frac{2}{j-i+1}$

2 choices: $z_i, z_j \Rightarrow$ comparison

$$\Pr(\text{compare } z_i, z_j) = \frac{2}{j-i+1}$$

$$\sum_{i,j} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2\left(\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots \frac{1}{n-i}\right)$$

$$\leq \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2\left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots \frac{1}{n}\right)$$

$$\leq \sum_{i=1}^{n} 2\ln(n) + 2$$

$$\leq 2n\ln(n) + 2n$$

$$= O(n\log n)$$

Useful Fact:

$$\sum_{j=1}^{n} \frac{1}{j} \leq \ln(n) + 1$$