Quick Sort and median selection

*Alon Efrat*

Based on slides courtesy of Piotr Indyk and Carola Wenk

---

**QuickSort**

example of the divide-and-concourse paradigm

- Sorts “in place” (no need for extra space). Like insertion sort, but not like merge sort.
- Very practical (with tuning).
Divide and conquer

Quick sort an \( n \)-element array:

1. **Divide:** Partition the array into two subarrays around a pivot \( x \) such that elements in lower subarray \( \leq x \leq \) elements in upper subarray.

   \[
   \begin{array}{c}
   \leq x \\
   x \\
   \geq x
   \end{array}
   \]

2. **Conquer:** Recursively sort the two subarrays.

3. **Combine:** Trivial.

   **Key:** Linear-time partitioning subroutine.

---

Partitioning subroutine

\[
\text{PARTITION}(A, p, q) \quad \triangleright A[ p \ldots q] \\
x \leftarrow A[ p] \quad \triangleright \text{pivot } = A[ p] \\
i \leftarrow p \\
\text{for } j \leftarrow p+1 \text{ to } q \\
\quad \text{do if } A[ j] \leq x \\
\quad \quad \text{then} \\
\qquad \iota \leftarrow i+1 \\
\qquad \text{exchange } A[ \iota] \leftrightarrow A[ j] \quad \triangleright \text{Now } A[\iota] > x \\
\quad \text{end if} \\
\text{end for} \\
\text{exchange } A[ p] \leftrightarrow A[ i] \\
\text{return } i
\]

*Invariant:*

\[
\begin{array}{c}
\leq x \\
\otimes
\geq x
\end{array}
\]

Running time = \( O(n) \) for \( n \) elements.
Example of partitioning

6 10 13 5 8 3 2 11

i   j

Example of partitioning

6 10 13 5 8 3 2 11

i   →  j
Example of partitioning

\[ \begin{array}{cccccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
\end{array} \]

\[ i \quad \longrightarrow \quad j \]

Example of partitioning

\[ \begin{array}{cccccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
6 & 5 & 13 & 10 & 8 & 3 & 2 & 11 \\
\end{array} \]

\[ i \quad \longrightarrow \quad j \]
Example of partitioning

\[\begin{array}{cccccccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
6 & 5 & 13 & 10 & 8 & 3 & 2 & 11 \\
\end{array}\]

\[\begin{array}{cccccccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
6 & 5 & 13 & 10 & 8 & 3 & 2 & 11 \\
\end{array}\]
Example of partitioning

Example of partitioning
Example of partitioning

6  10  13  5  8  3  2  11
6  5  13  10  8  3  2  11
6  5  3  10  8  13  2  11
6  5  3  2  8  13  10  11

Example of partitioning

6  10  13  5  8  3  2  11
6  5  13  10  8  3  2  11
6  5  3  10  8  13  2  11
6  5  3  2  8  13  10  11

\[i\] \[j\]
Pseudocode for quicksort

**QUICKSORT***(A, p, r)***

if \( p < r \)

then \( q \leftarrow \text{PARTITION}(*A, p, r*) \)

QUICKSORT(*A, p, q−1*)

QUICKSORT(*A, q+1, r*)

**Initial call:** QUICKSORT(*A, 1, n*)

---

Analysis of quicksort

- Assume all input elements are distinct.
- In practice, there are better partitioning algorithms for when duplicate input elements may exist.
- Let \( T(n) = \text{worst-case running time on an array of } n \text{ elements.} \)
Worst-case of quicksort

• Input sorted or reverse sorted.
• Partition around min or max element.
• One side of partition always has no elements.

\[ T(n) = T(0) + T(n - 1) + \Theta(n) \]
\[ = \Theta(1) + T(n - 1) + \Theta(n) \]
\[ = T(n - 1) + \Theta(n) \]
\[ = \Theta(n^2) \] \textit{(arithmetic series)}

Worst-case recursion tree

\[ T(n) = T(0) + T(n-1) + cn \]
Worst-case recursion tree

\[ T(n) = T(0) + T(n-1) + cn \]
Worst-case recursion tree

\[ T(n) = T(0) + T(n-1) + cn \]

\[ \Theta(1) \]
Worst-case recursion tree

\[
T(n) = T(0) + T(n-1) + cn
\]

\[
\Theta \left( \sum_{k=1}^{n} k \right) = \Theta(n^2)
\]

14.25
Best-case and almost best-case analysis

If we are lucky, **PARTITION** splits the array evenly:

\[ T(n) = 2T(n/2) + \Theta(n) \]
\[ = \Theta(n \lg n) \quad \text{(same as merge sort)} \]

What if the split is always \( \frac{1}{10} : \frac{9}{10} \) ?

\[ T(n) = T(\frac{1}{10}n) + T(\frac{9}{10}n) + \Theta(n) \]

What is the solution to this recurrence?

Analysis of “almost-best” case

\[ T(n) \]
Analysis of “almost-best” case

$cn \quad T(\frac{1}{10}n) \quad T(\frac{9}{10}n)$

Analysis of “almost-best” case

$cn \quad \frac{1}{10}cn \quad \frac{9}{10}cn$

$T(\frac{1}{100}n)T(\frac{9}{100}n) \quad T(\frac{9}{100}n)T(\frac{81}{100}n)$
Analysis of “almost-best” case

\[ \Theta(1) \leq T(n) \leq cn \log_{10/9} n + O(n) \leq 8cn \log_2 n \]

\[ O(n) \text{ leaves} \]
Randomized quicksort

How can find a pivot that guarantees partitions with good ratios for \( A[1..n] \)?

We say that \( q \) is a good pivot for if:

- at least 10% of the elements of \( A[1..n] \) are smaller than \( q \), and
- at least 10% of the elements of \( A[1..n] \) are larger than \( q \).

\[
\begin{align*}
10\% & \leq q \\
10\% & \geq q
\end{align*}
\]

Best pivot: Pick the median of \( A[1..n] \), as pivot.

Then the time would obey \( T(n) = cn + 2T(n/2) \)

Problem – need to work too hard to find the median (best pivot), so we will do with (only) a good pivot.

Finding a good pivot for \( A[1..n] \)

5-random-elements method:

- Pick the indices of 5 elements at random from \( A[1..n] \).
- For \( k = 1 \) to \( 5 \)
  \[
  X[k] = A[n \text{ rand}()]
  \]

\( A[1..n] \)

- Set \( q \) to be the median of \( X[1..5] \).
Finding a good pivot for $A[1..n]$

5-random-elements method: Pick 5 elements at random from $A[1..n]$, and set $q$ to be their median.

What is the probability that $q$ is not a good pivot?

- Let $S$ be the elements of $A[1..n]$ which are the 10% smallest.
- The probability that an element picked at random is in $S$ is 0.1.
- $q$ is in $S$ only if at least 3 of the 5 elements that we pick are in $S$.
- The probability that this happens is $0.1^5 + 5\cdot 0.1^4 \cdot 0.9 + 10\cdot 0.1^3 \cdot 0.9^2 = 0.00001 + 0.00045 + 0.00810 = 0.00856$
- This is also the probability that $q$ is in the 10% largest elements.
- In other words: with probability $\geq 98\%$, $q$ is a good pivot.

Randomized quicksort – cont
Finding good pivots

Putting it together, during QS, each time that we need to find a pivot, we use the “5 random elements” method.

With probability 98%, we find a good pivot.

The overall time that we spend on good partitions is much smaller than the time we spent on bad partitions.

(note – bad partitions are not harmful – they are just not helpful)

So the recursions formula $T(n) = cn + T(n/10) + T(n/9)\) still apply, leading to running time $O(n \log n)$.

This is expected running time – there is a chance that the actual running time is $\Theta(n^2)$, but the probability that it happens is very slim.
Quicksort in practice

• Quicksort is a great general-purpose sorting algorithm.
• Quicksort is typically over twice as fast as merge sort.
• Quicksort behaves well even with caching and virtual memory.

Median Selection

• (CLRS Section 9.2, page 185).
• For $A[1..n]$ (all different elements) we say that the rank of $x$ is $i$ if exactly $i-1$ elements in $A$ are smaller than $x$.
• In particular, the median is the $\lfloor n/2 \rfloor$-smallest.
• To find the median, we could sort and pick $A[\lfloor n/2 \rfloor]$ (taken $O(n \log n)$).
• We can do better.
Median Selection-cont

RS(A, p, r, i){
    //Randomize Selection: Returns i’st smallest element in A[p..r].
    //Assumption: Input is valid and elements are different.
    • If p==r return A[p]
    • q=PARTITION(A,p,r);
        • //Partition using the 5-random element method
        • k=q-p
        • If i==k+1 return A[q]
        • If i<k return RS(A, p, q-1, i) // Note the difference from QS
        • Else   return RS(A, q+1, r, i-k-1)
}

Time analysis

• Recall: With high probability, we pick a good pivot:
  • Not in the 10% smallest or largest:
  • Hence, we get rid of at least 10% of the elements of A
• So, T(n)=cn+T(0.9 n).
  • T(n)=c(n+0.9n+ 0.9^2n+0.9^3n+...) =
    cn(1+0.9+ 0.9^2+0.9^3+...) =
    cn(1/(1-0.9)) = O(n).
• So the expected time is linear. (yuppie)

As in the case of QS, partitions which are not good are not harmful,
just not helpful.