Data Structures

Quicksort
Quicksort

• To sort the subarray A[p . . r]:
  – **Divide:**
    • Partition A[p . . r] into A[p . . q − 1] and A[q + 1 . . r] such that
      – Each element in A[q + 1 . . r] is ≥ A[q].
  – **Conquer:**
    • Sort the two subarrays by recursive calls to quicksort.
  – **Combine:**
    • No work is needed to combine the subarrays, because they are sorted in place.
QuickSort

\textbf{quickSort} (A, p, r )

\textbf{if} (p < r) \textbf{then}

\hspace{1cm} q = \textbf{partition} (A, p, r )

\hspace{1cm} \textbf{quickSort} (A, p, q - 1)

\hspace{1cm} \textbf{quickSort} (A, q + 1, r )

\textbullet \textbf{Initial call} is quickSort (A, 1, n).
Partition

- All entries in $A[j .. r-1]$ are not yet examined.
- All entries in $A[i+1 .. j-1]$ are $> pivot$.
- All entries in $A[p .. i]$ are $\leq pivot$.
Partition

\[ i = p - 1 \]

\[ j = p \]

\[ \begin{array}{cccccccc}
2 & 8 & 6 & 3 & 5 & 1 & 7 & 4 \\
\end{array} \]
Partition

\[ i = p \]

\[ j = p + 1 \]
Partition

\[ i = p \]

\[ j = p + 2 \]
Partition

i = p

j = p + 3
Partition

\[ i = p + 1 \]

\[ j = p + 4 \]
Partition

\[ i = p + 1 \]

\[ j = p + 5 \]
Partition

\[ i = p + 2 \]

\[ j = p + 6 \]
Partition

\[ i = p + 2 \]

\[ j = p + 7 \]
Partition

\[ i = p + 2 \]
partition (A, p, r )
    i = p − 1
    for (j = p to r − 1) do
        if (A[j] ≤ A[r]) then
            i = i + 1
            exchange A[i] with A[j]
        exchange A[i + 1] with A[r]
    return i + 1

• **Complexity**: $\Theta(n)$ to partition an n-element subarray
Performance

• The running time of quicksort depends on the partitioning of the subarrays:
  – If the subarrays are balanced, then quicksort can run as fast as mergesort.
  – If they are unbalanced, then quicksort can run as slowly as insertion sort.
Worst Case

• Occurs when the subarrays are completely unbalanced every time.
  – When quicksort takes a sorted array as input.
• Have 0 elements in one subarray and n – 1 elements in the other subarray.
• The time complexity recurrence is

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

\[
= T(n - 1) + \Theta(n)
\]

\[
= \Theta(n^2)
\]

• Same running time as insertion sort.
Best Case

- Occurs when the subarrays are completely balanced every time.
- Each subarray has \( \leq n/2 \) elements.
- The time complexity recurrence is

\[
T(n) = 2T(n/2) + \Theta(n)
\]

\[
= \Theta(n \log n)
\]
Balanced partitioning

• Imagine that partition always produces a 9-to-1 split.

• The time complexity recurrence is

\[
T(n) \leq T(9n/10) + T(n/10) + \Theta(n) \\
= O(n \lg n).
\]

– Any split of constant proportionality will yield a recursion tree of depth \(\Theta(\lg n)\).

– Here we get a tree with \(\log_{10} n\) full levels and \(\log_{10/9} n\) levels that are nonempty.
Balanced partitioning
Intuition for the average case

- **Average** running time is much closer to the **best** case than to the worst case.
- There will usually be a **mix** of good and bad splits throughout the recursion tree.
- When the number of bad splits is bound, it doesn’t affect the asymptotic running time.
Summary

• **Worst-case** running time: $O(n^2)$.  
• **Expected** running time: $O(n \lg n)$.  
• **Constants** hidden in $O(n \lg n)$ are small.  
• Sorts **in place**.
Randomized Version

• Instead of assuming that all input permutations are equally likely, we use randomization.
  – Instead of using A[r] as the pivot, randomly pick an element from the subarray that is being sorted.

```plaintext
randomizedPartition (A, p, r )
i = random (p, r)
exchange A[r] with A[i]
return partition (A, p, r )

quicksort (A, p, r )
if (p < r) then
  q = randomizedPartition (A, p, r )
quicksort (A, p, q − 1)
quicksort (A, q + 1, r )
```
Average Case

- The **dominant cost** is partitioning.
  - Since partition removes the pivot-elements from future consideration, the total number of calls is $O(n)$.
- Let $X$ be the **total number of comparisons** in all partitions.
- We compute a bound on $X$:
  - Rename the elements of $A$ as $z_1, z_2, \ldots, z_n$, with $z_i$ being the $i$th smallest element.
  - Define the set $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$ to be the set of elements between $z_i$ and $z_j$, inclusive.
  - Each pair of elements is compared at most once,
    - Because elements are compared only to the pivot element, and then the pivot element is never in any later call to partition.
  - Let $X_{ij} = I\{z_i$ is compared to $z_j\}$. 

Average Case

• Since each pair is compared at most once, the total number of comparisons is:

\[ X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \]

• Take expectations of both sides:

\[
E[X] = E \left[ \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \right] \\
= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \\
= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\} \]
Average Case

• The **probability** that $z_i$ is compared to $z_j$ is the probability that either $z_i$ or $z_j$ is the first element chosen in $Z_{ij}$.
  - Since once a pivot $x$ is chosen such that $z_i < x < z_j$, then $z_i$ and $z_j$ will never be compared at any later time.
• There are $j-i+1$ elements in $Z_{ij}$, and pivots are chosen randomly and independently.
• Thus, the probability that any particular one of them is the first one chosen is $1/(j-i+1)$.
• Therefore,

\[
\Pr\{z_i \text{ is compared to } z_j\} = \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
= \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\} + \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
= \frac{1}{j-i+1} + \frac{1}{j-i+1} \\
= \frac{2}{j-i+1}.
\]
Average Case

- Substituting into the equation for $E[X]$: 

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \quad (k = j-i)$$

$$< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \quad (H_n = \sum_{k=1}^{n} \frac{1}{k})$$

$$= \sum_{i=1}^{n-1} O(\lg n) \quad (\lim_{n \to \infty} H_n - \ln(n) = \gamma)$$

$$= O(n \lg n).$$

- So the expected running time of quicksort, using randomizedPartition is $O(n \lg n)$. 
Order Statistic

• The i-th order statistic of a set of n elements is the i-th smallest element
  – The minimum is the first order statistic (i = 1)
  – The maximum is the n-th order statistic (i = n)

• A median is the "halfway point" of the set
  – When n is odd, the median is unique, at i = (n + 1)/2.
  – When n is even, there are two medians, at i = n/2 (the lower median) and i = n/2 + 1 (the upper median)

• The selection problem:
  – Input: A set A of n (distinct) numbers and a number i, with 1 ≤ i ≤ n.
  – Output: The element x ∈ A that is larger than exactly i - 1 other elements of A.

• The selection problem can be solved in O(n lg n) time (sort and pick the ith element)

• We’ll see two algorithms
  – O(n) bound on the running time in the average case
  – O(n) running time in the worst case
Minimum and Maximum

- Minimum
- Maximum
- Simultaneous
- $O(n)$

```
minimum(A)
min \leftarrow A[1]
for i \leftarrow 2 to length[A] do
  if min > A[i] then
    min \leftarrow A[i]
return min
```
Selection in Expected Linear Time

```plaintext
RANDOMIZED-SELECT(A, p, r, i)
1 if p = r
2 then return A[p]
3 q ← RANDOMIZED-PARTITION(A, p, r)
4 k ← q - p + 1
5 if i = k    → the pivot value is the answer
6 then return A[q]
7 elseif i < k
8 then return RANDOMIZED-SELECT(A, p, q - 1, i)
9 else return RANDOMIZED-SELECT(A, q + 1, r, i - k)
```
Selection in Expected Linear Time

\[ X_k = \{ \text{the subarray } A[p \ldots q] \text{ has exactly } k \text{ elements} \} \]

\[ E[X_k] = \frac{1}{n} . \]

\[ T(n) \leq \sum_{k=1}^{n} X_k \cdot (T(\max(k-1, n-k)) + O(n)) \]

\[ = \sum_{k=1}^{n} (X_k \cdot T(\max(k-1, n-k)) + O(n)) . \]

We solve the recurrence by substitution

Assume that \( T(n) \leq cn \) and that \( T(n) = 1 \) for small \( n \)

\[ E[T(n)] \leq \frac{2}{n} \sum_{k=[n/2]}^{n-1} ck + an \]

\[ = \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{[n/2]-1} k \right) + an \]

\[ = \frac{2c}{n} \left( \frac{(n-1)n}{2} - \frac{([n/2]-1)[n/2]}{2} \right) + an \]

\[ \leq \frac{2c}{n} \left( \frac{(n-1)n}{2} - \frac{(n/2-2)(n/2-1)}{2} \right) + an \]

\[ = \frac{2c}{n} \left( \frac{n^2 - n}{2} - \frac{n^2/4 - 3n/2 + 2}{2} \right) + an \]

\[ = \frac{c}{n} \left( \frac{3n^2}{4} + \frac{n}{2} - 2 \right) + an \]

\[ = c \left( \frac{3n^2}{4} + \frac{1}{2} - \frac{2}{n} \right) + an \]

\[ \leq \frac{3cn}{4} + \frac{c}{2} + an \]

\[ = cn - \left( \frac{cn}{4} - \frac{c}{2} - an \right) . \]

Which is at most \( cn \) for \( c > 4a \) and 

\[ n \geq \frac{c/2}{c/4 - a} = \frac{2c}{c - 4a} . \]
Selection in Worst-Case Linear Time

1. **if** \( n = 1 \) **then**
   
   Returns the only input value.

2. **Else**

   1. Divide the \( n \) elements of the input array into \( \lceil n/5 \rceil \) groups of 5 elements each and at most one group made up of the remaining \( n \) mod 5 elements.
   2. Find the median of each of the \( \lceil n/5 \rceil \) groups by first insertion sorting the elements of each group and then picking the median from the sorted list of group elements.
   3. Recursively find the median \( x \) of the \( \lceil n/5 \rceil \) medians found in step 2
   4. Partition the input array around the median-of-medians \( x \).
   5. Let \( k \) be the number such that \( x \) is the \( k \)-th smallest element.
   6. If \( i = k \)
      
      return \( x \)
   7. Else
      
      Recursively find the \( i \)-th smallest element on the low side if \( i < k \), or the (\( i - k \))th smallest element on the high side if \( i > k \).
Selection in worst-case linear time

After each partition we rid of at least $3\left\lfloor \frac{n}{5} \right\rfloor - 2 \geq \frac{3n}{10} - 6$ elements.

We solve the recurrence by substitution

Assume that $T(n) \leq cn$ and that $T(n) = 1$ for small $n$

Which is at most $cn$ for $n \geq 140$ and $c \geq 20\alpha$