Chapter 7. Quicksort

- Worst-case running time is $\Theta(n^2)$
- On average: $\Theta(n \lg n)$ and the constant factors that are hidden are quite small
- Sorts in place

7.1 Description of Quicksort

- Divide-and-conquer strategy:

  Divide: Partition the array $A[p \ldots r]$ into two subarrays $A[p \ldots q-1]$ and $A[q+1 \ldots r]$ such that:

  $$\begin{cases} \text{Each element in } A[p \ldots q-1] \leq A[q] \\ \text{Each element in } A[q+1 \ldots r] \geq A[q] \end{cases}$$

  Conquer: Sort the two subarrays $A[p \ldots q-1]$ and $A[q+1 \ldots r]$ by recursive calls to quicksort

  Combine: Since the subarrays are sorted in place, no work is need to combine them: the entire array $A[p \ldots r]$ is now sorted.

```
Quicksort (A, p, r) {
    if (p < r) {
        q = Partition (A, p, r)
        Quicksort (A, p, q-1)
        Quicksort (A, q+1, r)
    }
}
```

Notes:

To partition an entire array:

Quicksort (A, 1, Length(A))

7.1.1 Partitioning the array

- The key to the algorithm is the partition procedure

```
Partition (A, p, r) {
    x = A[r] /* Pivot element */          } Θ(1)
    i = p - 1                             }
    for (j = p to r-1) {                  ——— Θ(n)
        if (A[j] ≤ x) {                   }
            i = i+1;                      } Θ(1)
            Exchange A[i] ↔ A[j]          }
        }
    }
    exchange A[i+1] ↔ A[r]                } Θ(1)
    return i+1
}
```

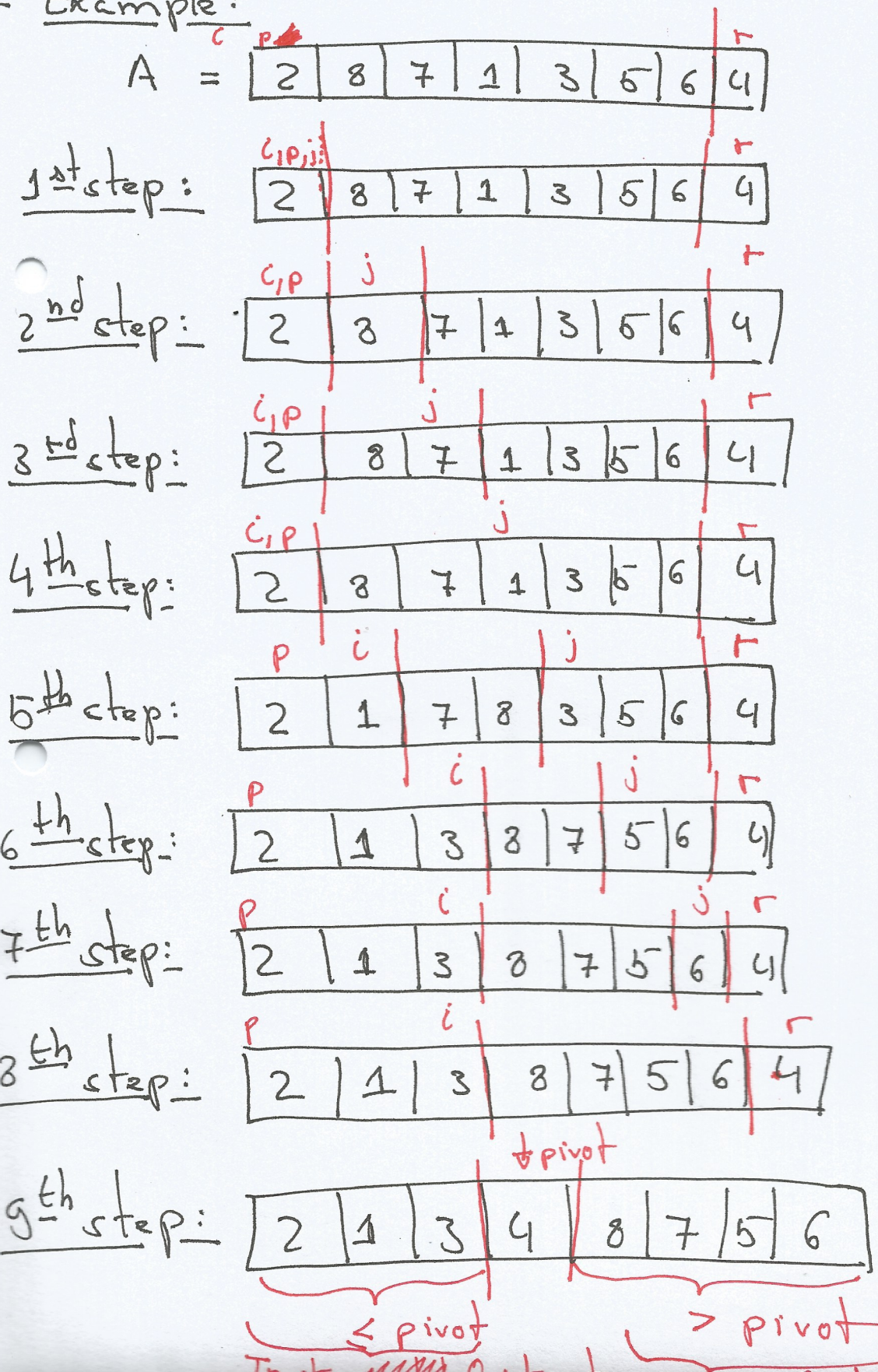
Total: $\Theta(1) + \Theta(n).\Theta(1) + \Theta(1)$ (notation abuse)

$= \Theta(1) + \Theta(n) + \Theta(1)$

$= \Theta(n)$

- The Partition procedure always selects an element $x = A[r]$ as a pivot element around which to partition the subarray $A[p \ldots r]$

- Example:



At the beginning of each iteration of the Partition loop we have:

1. $\forall_{k \in [p,i]} A[k] \leq q$ ← pivot
2. $\forall_{k \in [i+1, j-1]} A[k] > q$ ← pivot
3. $\forall_{k \in [j, r-1]}$ "Unknow"
4. $\forall_{k=r} A[k] = q$ ← pivot

Question: What is the total running time of the Partition procedure?

- For each element of the array we need to find the correct place "around" the pivot
- There are $n-1$ elements that we always need to consider, therefore $\Theta(n)$
- We also employ some constant time calculations before, during and after the partition cycle
- Total time: $\Theta(1) + \overbrace{\Theta(n) \cdot \Theta(1)}^{\text{notation abuse}} + \Theta(1)$

$$= \Theta(1) + \Theta(n) + \Theta(1) = \Theta(n)$$

7.2 Performance of Quicksort

Question:

How can we analyse the performance of quicksort?

- **Q:** What is the worst-case scenario you can envision for the partition procedure?

  ↳ What if the partitioning "fails"? I.e. the pivot chosen is larger than all the other elements [unbalanced partitioning idea]

  ↳ This would produce one subproblem with $n-1$ elements and one with 0 elements

- **Q:** Worse yet: what happens when this unbalanced partitioning arises in each recursive call?

- We are dealing with a divide-and-conquer algorithm therefore we can describe the total cost by a recurrence:

$$T(n) = \underbrace{T(n-1)}_{\text{time to process the 'left' partition that has } n-1 \text{ elements}} + \overbrace{T(0)}^{\text{time to process the 'right' partition, that has zero elements}} + \underbrace{\Theta(n)}_{\text{The partitioning cost of calculating the left and right partitions}}$$

$T(n) = T(n-1) + \underbrace{T(0)}_{\Theta(1)} + \Theta(n)$

$= T(n-1) + \Theta(1) + \Theta(n)$

We can ignore this since we are interested in the asymptotic behaviour

$= T(n-1) + \Theta(n)$

Q: What does this mean?

Notes:

Arithmetic Series:

$$\sum_{k=1}^{n} k = \frac{1}{2} n(n+1)$$

- Each level will have an $\Theta(n)$ cost
- Until eventually there are no more partitions to be created
- E.g.: $T(5) = T(4) + \Theta(n)$

$T(4) = T(3) + \Theta(n)$

$T(3) = T(2) + \Theta(n)$

$T(2) = T(1) + \Theta(n)$

$T(1) = T(0) + \Theta(n)$ (recursion will stop)

$T(0) = \Theta(1)$

From a different but equivalent perspective:

$T(5) = T(4) + \Theta(n) = T(3) + \Theta(n) + \Theta(n) = T(2) + \Theta(n) + \Theta(n) + \Theta(n)$

$= T(1) + \Theta(n) + \Theta(n) + \Theta(n) + \Theta(n) =$

$= T(0) + \underbrace{\Theta(n) + \Theta(n) + \Theta(n) + \Theta(n) + \Theta(n)}$

$$\sum \Theta(n) = \sum_{k=1}^{5} \Theta(k) = \Theta\left(\sum_{k=1}^{5} k\right)$$

~~Geometric~~ Arithmetic Series

- Back to the recurrence:

$T(n) = T(n-1) + \Theta(n)$

$= \sum_{k=1}^{n} k = \frac{1}{2} n(n+1) = \frac{1}{2}(n^2 + n) \Rightarrow \Theta(n^2)$

Quicksort perfor

Q: What is the best-case scenario you can envision for the partition procedure

- Ideally, we should be able to split the array evenly:
  - Left partition: $n/2$ elements
  - Right partition: $n/2 - 1$ elements

  } Well, almost evenly ;)

- This would result in a recurrence:

$$T(n) = 2T(n/2) + \Theta(n)$$

Case 2 of the master theorem:

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

Notes:

Master Theorem:

Case 1:
$f(n) = O(n^{\log_b a - \epsilon}) \Rightarrow T(n) = \Theta($

Case 2:
$f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta($

Case 3:
$f(n) = \Omega(n^{\log_b a + \epsilon}) \Rightarrow T(n) = \Theta($

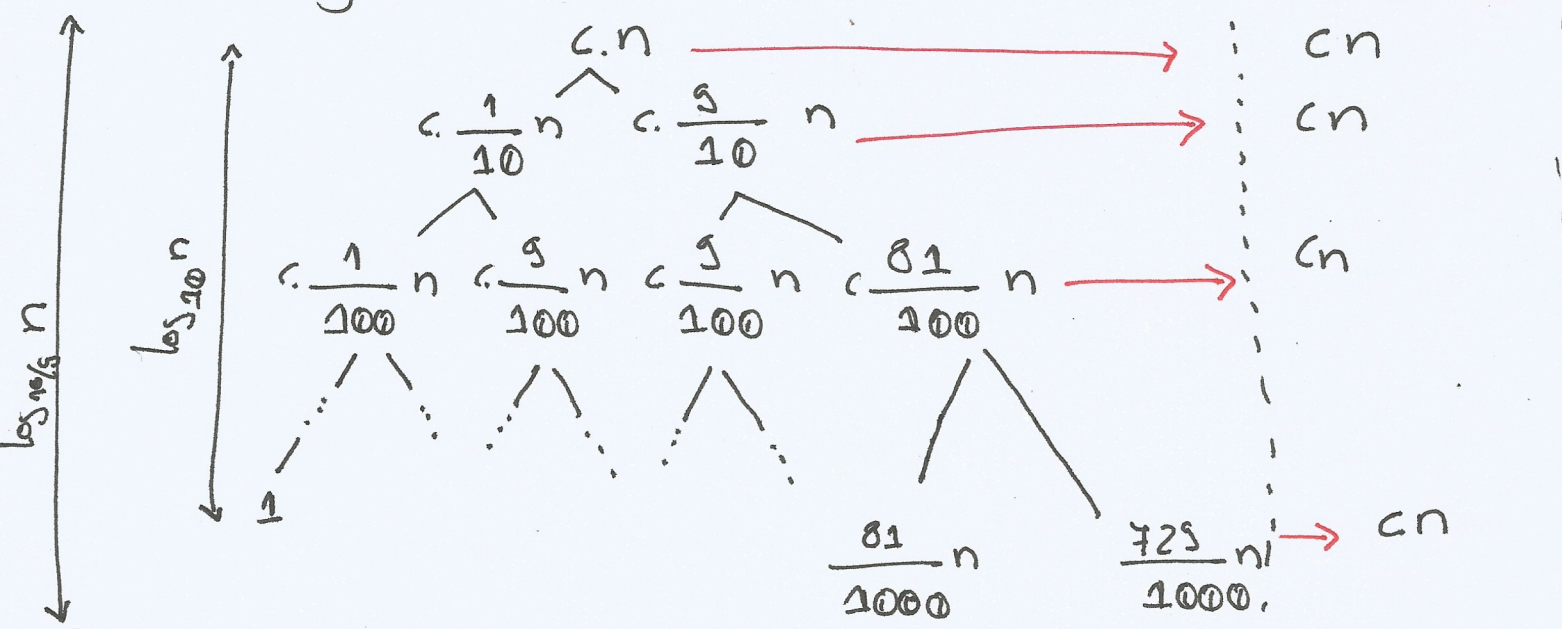
Q: What if we have a different partitioning scheme laying somewhere in the middle of the worst-case and the best-case scenarios?

- Suppose, for example, that the partitioning scheme always produces a 9-to-1 split (this seems very unbalanced)
- Then the recurrence would be:

$$T(n) = T\left(\frac{9}{10}n\right) + T\left(\frac{n}{10}\right) + \Theta(n)$$

$$= T\left(\frac{9}{10}n\right) + T\left(\frac{n}{10}\right) + cn$$

- Drawing the recurrence tree:



- How long will be the shortest path?

Depth	Input Size
0	$n = n/10^0$
1	$n/10 = n/10^1$
2	$n/100 = n/10^2$
3	$n/1000 = n/10^3$
⋮	
δ	$n/10^d$

If we assume that n is a power of 10 then for some d the ratio $\frac{n}{10^d}$ will be 1, in other words:

$$d = \log_{10} n$$

- How long will be the longest path?

Depth	Input Size
0	$n = (9/10)^0 . n$
1	$9/10 . n = (9/10)^1 . n$
2	$81/100 . n = (9/10)^2 . n$
3	$729/1000 . n = (9/10)^3 . n$
⋮	⋮
d	$(9/10)^d . n$

If we assume that n is a power of $(9/10)$ then for some d the ratio $\frac{n}{(10/9)^d}$ will be 1

In other words:

$$d = \log_{10/9} n$$

- Minimum tree depth: $\log_{10} n = \Theta(\log n)$
- Maximum tree depth: $\log_{10/5} n = \Theta(\log n)$
- Notice that every level of the tree has cost cn
  - This happen until (including) the boundary condition of the shortest path is reached
  - From this point on we will gradually start to lose elements
  - Thus the overall cost per level will be $\leq cn$
- This means that the total ~~recurrence~~ cost of the tree will be:

  maximumNumberOfLevels x costperLevel

  $= \log_{10/5} n \times cn =$

  $= \Theta(\log n) \times \Theta(n) = \Theta(n \log n)$

∴ Even with an apparent unbalanced partition the algorithm still runs in $\Theta(n \log n)$ time

Observation:

A good way to avoid the worst-case scenario described in page 5 is to "randomize" the array before ~~part~~ performing Quicksort

→ The probability of $\Theta(n^2)$ will be very small for large arrays