csce350 — Data Structures and Algorithms
Fall 2019 — Lecture Notes: Decrease and Conquer

This document contains slides from the lecture, formatted to be suitable for printing or individual reading, and with occasional supplemental explanations added. It is intended as a supplement to, rather than a replacement for, the lectures themselves — you should not expect the notes to be self-contained or complete on their own.

1 Introduction

Decrease and conquer algorithms are based on solving a single smaller subproblem, and then extending that solution to apply to the original problem.

Three flavors:

- Decrease by a constant.
- Decrease by a constant factor.
- Variable size decrease.

2 Integer powers

Recall the **integer powers** problem: Given two positive integers a and n, compute a^n.

Facts:

- If n is even, then $a^n = a^{n/2} \cdot a^{n/2} = \left(a^{n/2}\right)^2$.
- If n is odd, then $a^n = a^{\lfloor n/2 \rfloor} \cdot a^{\lfloor n/2 \rfloor} \cdot a = \left(a^{\lfloor n/2 \rfloor}\right)^2 \cdot a$.
- If $n = 1$, $a^n = a$.

3 Integer powers

```
INTEGERPOWER(a, n)
  if n = 1 then
    return a
  else if n is even then
    t ← INTEGERPOWER(a, n/2)
    return t · t
  else
    t ← INTEGERPOWER(a, ⌊n/2⌋)
    return t · t · a
  end if
```

4 Integer powers analysis

$M(n) = M(n/2) + 2$

$M(n) \in \Theta(\log n)$

5 Insertion sort

Idea:

- Sort the first $n-1$ elements.
- Find the correct home for the final element and insert it there.

6 Insertion sort

```
INSERTIONSORT(A[0, ..., n − 1])
  if n > 1 then
    INSERTIONSORT(A[0, ..., n − 2])
    v ← A[n − 1]
    j ← n − 2
    while j ≥ 0 and A[j] > v do
      A[j + 1] ← A[j]
      j ← j − 1
    end while
    A[j + 1] ← v
  end if
  return
```

7 Insertion sort analysis

Worst case:

$$C(n) = C(n-1) + \Theta(n) = \Theta(n^2)$$

Best case:

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

8 Binary search

Problem: Array search

- **Input:** A **sorted** array $A[0, \ldots, n-1]$ and a search key K.
- **Output:** An index i such that $A[i] = K$, or -1 if there is no such index.

Idea: Cut the range of possible locations for K in half each time.

9 Binary search

```
BINARYSEARCH(A[0, ..., n − 1], ℓ, u, K)
  if ℓ > u then
    return −1
  end if
  m ← ⌊(u − ℓ + 1)/2⌋
  if K = A[m] then
    return m
  else if K > A[m] then
    return BINARYSEARCH(A, m + 1, u, K)
  else
    return BINARYSEARCH(A, ℓ, m − 1, K)
  end if
```

See also: Iterative version, page 136.

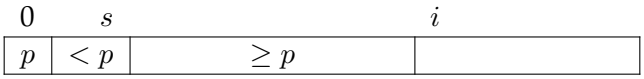
10 Partitioning an array

- **Input:** An array $A[0, \ldots, n-1]$.
- **Output:** Call $p \leftarrow A[0]$ the **pivot** element. Re-arrange A so that
  - Elements left of p are less than p.
  - Elements right of p are greater than or equal to p.

  Return the final index of the pivot.

11 Partitioning algorithm: Idea

Scan left-to-right, keeping this structure:



- $A[0]$: pivot
- $A[1, \ldots, s]$: less than pivot
- $A[s+1, \ldots, i-1]$: greater than or equal to pivot
- $A[i, \ldots, n-1]$: not checked yet

12 Partition

```
PARTITION(A[0, ..., n − 1])
  p ← A[0]
  s ← 0
  for i ← 1, ..., n − 1 do
    if A[i] < p then
      Swap A[s + 1] and A[i].
      s ← s + 1
    else
      Do nothing.
    end if
  end for
  Swap A[0] and A[s].
  return s
```

13 The selection problem

The objective of the **selection problem** is to find the kth smallest element in an array.

- **Input:** An array A of n distinct comparable elements, and an integer k.
- **Output:** An element a from A that is greater than exactly k other elements of A.

For example:

- $k = 0$: smallest
- $k = n - 1$: largest
- $k = \lfloor n/2 \rfloor - 1$: median

14 QuickSelect

```
QUICKSELECT(A[0, ..., n − 1], k)
  s ← PARTITION(A[0, ..., n − 1])
  if s = k then
    return A[s]
  else if k < s then
    return QUICKSELECT(A[0, ..., s − 1], k)
  else
    return QUICKSELECT(A[s + 1, ..., n − 1], k − (s + 1))
  end if
```