ALGORITHMS AND DATA STRUCTURES

SELECTION ALGORITHMS

THE PROBLEM OF SELECTING

- We're going to address the problem of selecting i-th extreme (largest or smallest) element from a set of *N* different numbers
- We're going to assume that all elements in the set are different

THE PROBLEM OF SELECTING

- Formally, the selection problem is stated as follows
  - **Input**: A set of N (different) numbers $A = \{a_1, \ldots, a_N\}$ and an integer i, with $1 <= i <= N$
  - **Output**: The element x in A that is larger than exactly i - 1 other elements of the set A. The item x is called the i-th *order statistic*

SELECTION ALGORITHMS

- A *selection algorithm* is an algorithm for finding the k-th smallest number in a list or array; such a number is called the k-th order statistic.
- The complexity of these algorithms depends on whether the input array is sorted or not

EXAMPLES OF ORDER STATISTICS

- For A and $1 <= k <= N$, we typically want to find the minimum ($k = 1$), the maximum ($k = N$), and the median ($k = N / 2$)

EXAMPLES OF ORDER STATISTICS

- For A and 1 <= k <= *N*, we typically want to find the minimum (k = 1), the maximum (k = *N*), and the median (k = *N* / 2)
- The simplest case of a selection algorithm is indeed trying to find the minimum or the maximum item in an unsorted array
- It's easy to design an algorithm with worst-case linear time. *Finding the extreme elements is an easy task*

EXAMPLES OF ORDER STATISTICS

- For A and 1 <= k <= *N*, we typically want to find the minimum (k = 1), the maximum (k = *N*), and the median (k = *N* / 2)
- The simplest case of a selection algorithm is indeed trying to find the minimum or the maximum item in an unsorted array
- It's easy to design an algorithm with worst-case linear time. *Finding the extreme elements is an easy task*
- On the contrary and intuitively, finding the median will be the hardest case of all order statistics

STRATEGIES FOR SELECTION ALGORITHMS

1. Using sorting algorithms
2. Partitioning the array
3. Resorting to data structures

PARTITIONING ARRAYS

- Using a technique similar to binary search [partitioning], it is possible to obtain linear-time algorithms
- The idea is of course to partition the array and using some criteria that helps "guessing" its k-th order statistic
- We'll study algorithms that use this idea when revising **fast** sorting algorithms and their variations to solve the selection problem

USING DATA STRUCTURES

- To find an order statistic in sublinear time is to store the data in an organized fashion
- We will explore two options in this course:
  - Lookup tables: hash tables, uniformly distributed arrays [bucket sort]
  - Tree-based data structures: binary trees and binary heaps

SELECTION BY SORTING

- How can we solve the selection problem resorting to the algorithms we know so far for sorting and searching?

SELECTION BY SORTING

- How can we solve the selection problem resorting to the algorithms we know so far for sorting and searching?
- *By sorting the array then selecting the desired element, selection can be reduced to sorting*

SELECTION BY SORTING

- How can we solve the selection problem resorting to the algorithms we know so far for sorting and searching?
- *By sorting the array then selecting the desired element, selection can be reduced to sorting*
- This is inefficient for selecting only one order statistic, but efficient when many selections are needed

SELECTION BY SORTING

- How can we solve the selection problem resorting to the algorithms we know so far for sorting and searching?
- *By sorting the array then selecting the desired element, selection can be reduced to sorting*
- This is inefficient for selecting only one order statistic, but efficient when many selections are needed
- Rather than sorting the whole array, one can instead use partial sorting to select the k smallest or k largest elements

SORTING BY SELECTION

- How can we use the selection algorithms in order to solve the problem of sorting?

SORTING BY SELECTION

- How can we use the selection algorithms in order to solve the problem of sorting?
- *One can incrementally sort by repeated selection*

SORTING BY SELECTION

- How can we use the selection algorithms in order to solve the problem of sorting?
- *One can incrementally sort by repeated selection*
- In the extreme, a fully sorted array allows $O(1)$ selection. Further, compared with first doing a full sort, incrementally sorting by repeated selection amortizes the sorting cost over multiple selections

EFFICIENCY OF SELECTION ALGORITHMS

- The table below lists some familiar selection algorithms
- Some we examined today, some we will analyze later on

```
Algorithm/Data structure                Running time
========================                =============
Slow sorting                            O(N^2)
Partial sorting                         O(k*N)
Partitioning                            O(N)
Lookup tables                           O(sqrt(N))
Binary tree/heap                        O(lg N)
```

EFFICIENCY OF SELECTION ALGORITHMS

```
FUNCTION slow_select:
   INPUT: integer array A[n], integer k
   OUTPUT: k-th order statistic A[k]
   USAGE: slow_select(A, k)
BEGIN
   FOR i: 1, k
    minIndex, minValue = i, A[i]
    FOR j: i+1, n
        IF A[j] < minValue THEN
          minIndex, minValue = j, A[j]
          swap(A[i], A[minIndex])
        END
    END
  END
  RETURN A[k]
END // slow_select
```