CS320: Honors Option

Elita Danilyuk

What did the honors option consist of?

Diving further into the courses textbook to study some of the other advanced algorithms from chapters not covered in the regular coursework of CS320.

07

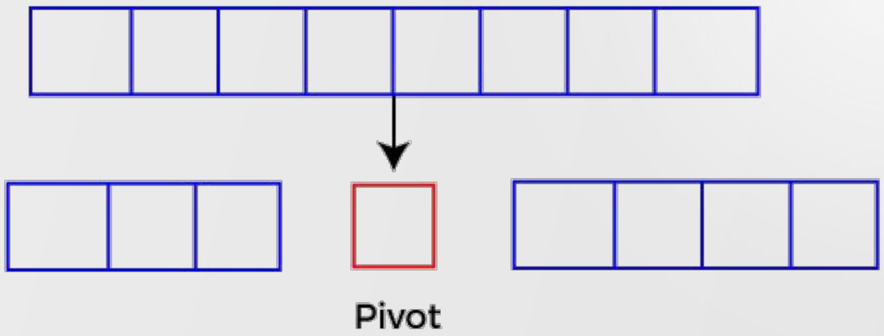
Chapter

Quicksort

Overview: Quicksort

Quicksort, on an input array of *n* numbers, has a worst-case running time of Θ(n^2).

- Often best practical choice for sorting
  - On average its expected running time is Θ(n log n)
  - Works well in virtual-memory environments



Section 7.1

Description of quicksort

- applies the divide-and-conquer method
  - partitions an array into two parts then sorts the parts independently

The partitioning process rearranges the array to make the following 3 conditions hold:

1. The entry A[j] is in its final place in the array, for some j
2. Each element of A[p … q - 1] ≤ A[q]
3. Each element of A[q + 1 … r] ≥ A[q]

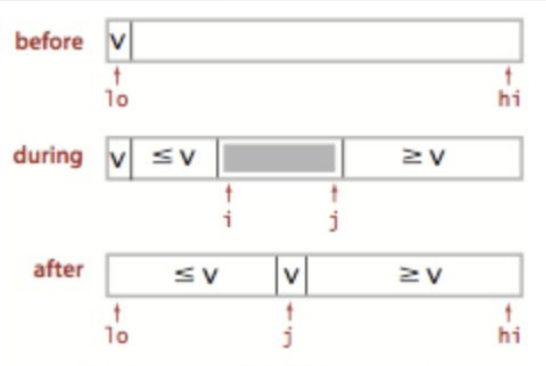
Then, we recursively apply the method to the subarrays

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```

Section 7.1 (Cont.)

PARTITION (A, 1, A.*length*)

- Selects an element *x* = A[*r*] as a ***pivot*** element to partition the subarray A[p…r] around
- The program partitions the array into 4 (possibly empty) regions



Quicksort partitioning overview

Loop invariant of *for* loop (lines 3-6):

1. If p ≤ k ≤ i, then A[k] ≤ x.
2. If i + 1 ≤ k ≤ j - 1, then A[k] > x.
3. If k = r, then A[k] = x.

PARTITION(A, p, r)

```
1  x = A[r]
2  i = p - 1
3  for j = p to r - 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Section 7.2

Performance of quicksort

Worst-case partitioning

- Occurs when partitioning creates one subproblem with n - 1 elements and the other with 0 elements
- If it is maximally unbalanced at each recursive level, the running time is $\Theta(n^2)$
  - This running time occurs when an input array is already completely sorted

Best-case partitioning

- When the partitioning produces the most even split possible, each subproblem is no more than n/2
  - If it is equally balanced then the running time is $\Theta(n \log n)$

09

Chapter

Medians & Order Statistics

Overview: Medians & Order Statistics

Quick vocabulary

- *i*th **order statistic**: *i*th smallest element of a set of *n* elements
- **minimum**: the first order statistic ($i = 1$)
- **maximum**: the *n*th order statistic ($i = n$)
- **median**: the "halfway point"
  - The textbook considers the median to refer to the lower median ($i = \lfloor (n + 1) / 2 \rfloor$)

Description of SELECTION

- Given a set A of *n* distinct numbers and a number *i*, $1 \le i \le n$, the **selection problem** computes the *i*th **order statistic** of A
  - **Input**: a set of *A* of *n* (distinct) numbers and an integer *i*, $1 \le i \le n$
  - **Output**: The element $x \in A$ that is larger than exactly $i - 1$ other elements of *A*

Section 9.1

The problem of selecting the minimum and maximum of a set of elements

MINIMUM (A)

- *n* - 1 comparisons is necessary to determine the minimum
  - The same is true to compute the maximum
- MINIMUM is optimal with respect to the number of comparisons performed

```
MINIMUM(A)
min = A[1]
for i = 2 to A.length
    if min > A[i]
        min = A[i]
return min
```

Section 9.1 (Cont.)

Simultaneous minimum and maximum

- Because it takes n - 1 comparisons for each the minimum and maximum, it takes 2n - 2 comparisons total
- The above can actually be done in at most $3 * \lfloor n / 2 \rfloor$ by maintaining the minimum and maximum elements and process
  the elements in pairs
  - The first pair is compared with each other
  - After that, the smaller is compared with the current minimum and the larger is compared to the current maximum

Section 9.2

A divide-and-conquer algorithm for the selection problem (RANDOMIZED-SELECT) that returns the *i*th smallest element of the array A[*p…r*]

- Like the quicksort algorithm previously described, the input array is partitioned recursively

```
RANDOMIZED-SELECT(A, p, r, i)
if p == r
    return A[p]
q = RANDOMIZED-PARTITION(A, p, r)
k = q − p + 1
if i == k               // the pivot value is the answer
    return A[q]
elseif i < k
    return RANDOMIZED-SELECT(A, p, q − 1, i)
else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

- Then, unlike quicksort, RANDOMIZED-SELECT works on only one side of the partition
    - Thus, the running time of RANDOMIZED-SELECT is $\Theta(n)$ (not $\Theta(n \log n)$)

Section 9.3

Selection in worst-case linear time (SELECT)

- *Guarantees* a good split upon partitioning the input array
- Uses the PARTITION from quicksort, previously discussed, but modified to take the element to partition around, the **pivot**, as an input parameter
- SELECT determines the ith smallest of an input array of $n > 1$ distinct elements

Section 9.3 (Cont.)

1. Divide the n elements of the input array into $\lfloor n / 5 \rfloor$ groups of 5 elements
   - The number of elements in the last group $\leq$ 5 elements
2. Find the median of each of the $\lceil n / 5 \rceil$ groups
   - Insertion-sort the elements of each group and find its median
3. Use SELECT recursively to 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 using the modified version of PARTITION (with x as the input parameter for the pivot)
5. If $i = k$: return x (the median-of-medians), else:
   i. if $i < k$: call SELECT recursively on the low side of the partition
   ii. else if $i > k$ call SELECT recursively on the high side of the partition

THANK YOU!