JOHNS HOPKINS UNIVERSITY | Engineering for Professionals

Partition and Selection

Foundations of Algorithms

Guven

JOHNS HOPKINS WHITING SCHOOL of ENGINEERING

Reading Assignment

- Cormen, Chapter 7, 9

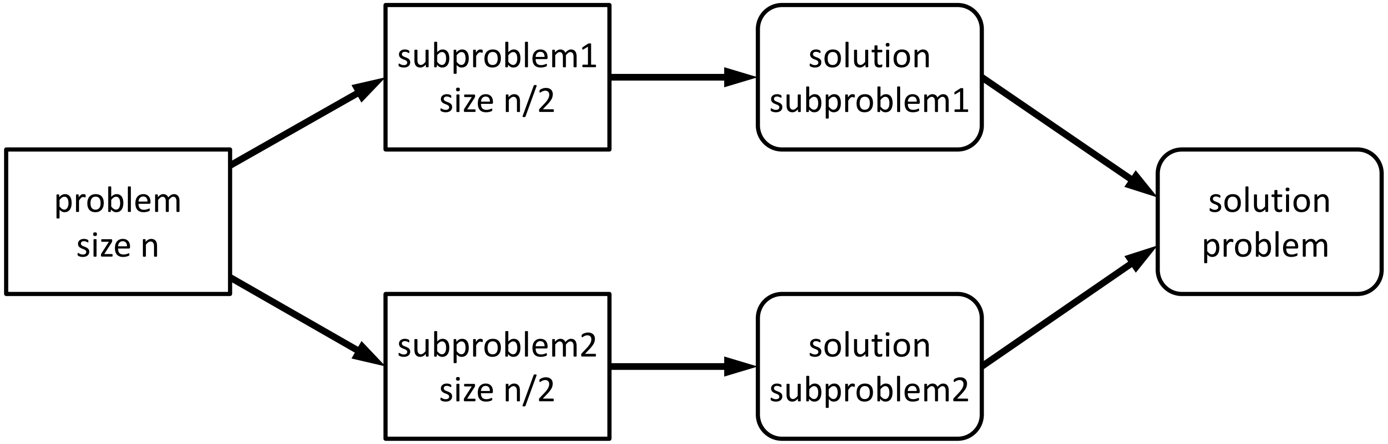
Outline

- Divide and Conquer
- Partition
- Selection
- Quickselect
- Deterministic Selection

Divide and Conquer

- Algorithm design paradigm:
  1. Divide instance of the problem into two or more smaller instances (disjoint)
  2. Solve smaller instances recursively
  3. Obtain solution to original (larger) instance by combining these (smaller) solutions
- Multi-branched recursion
  - e.g. Fibonacci sequence: $F_n = F_{n-1} + F_{n-2}$ (true benefit DC?)
  - e.g. Fast Fourier Transform (true benefit DC)
  - Note, in practice recursion is avoided by direct iteration

Divide and Conquer Paradigm



- Divide and conquer ⟷ e.g. Hadoop MapReduce
  - Not overlapping solutions
  - Not dynamic programming where the previous results are used in the next iteration
  - Each subproblem has an independent solution

Fibonacci Generation

- $F_0=0$, $F_1=1$, and $F_n=F_{n-1}+F_{n-2}$ $\forall$ $n\geq 2$
  - $\langle 0, 1, 1, 2, 3, 5, 8, 13, 21, .. \rangle$
- A recursive solution

```
def fibr(n):
    if n < 2:
        return n
    else:
        return fibr(n-1)+fibr(n-2)
```

- Time complexity: $O(2^n)$
  - `fibr` looks like a divide and conquer approach
  - But it is not. Why? Because the solutions are overlapping

Quicksort

```
def quicksort(A,p,r):
  if p<r:
    q=partition(A,p,r)  # divide
    quicksort(A,p,q-1)  # T(n/2)
    quicksort(A,q+1,r)  # T(n/2)

def partition(A,p,r):
  x=A[r]; i=p-1
  for j in range(p,r):  # Takes (r-p) operations
    if A[j]<=x:
      i+=1
      A[i],A[j]=A[j],A[i]   # swap
  A[i+1],A[r]=A[r],A[i+1]   # swap the pivot
  return i+1
```

- Quicksort is also called partition-exchange sort algorithm

Quicksort Example

i	***j,p***								***r***
	1	2	3	4	5	4	3	2	3
	i,p	***j***							***r***
	1	2	3	4	5	4	3	2	3
	p	***i***	***j***						***r***
	1	2	3	4	5	4	3	2	3
	p		***i***	***j***					***r***
	1	2	3	4	5	4	3	2	3
	p		***i***		***j***				***r***
	1	2	3	4	5	4	3	2	3
	p		***i***			***j***			***r***
	1	2	3	4	5	4	3	2	3
	p		***i***				***j***		***r***
	1	2	3	4	5	4	3	2	3
	p			***i***				***j***	***r***
	1	2	3	4	5	4	3	2	3
	1	2	3	3	5	4	3	2	4

In-class Exercise

- A=[10,2,8,4,6,1,2,2,5]
- Write down the partitions p, r, A at the end of first and second iterations
- Answer
  - 0 8 [2, 4, 1, 2, 2, 5, 10, 6, 8]
  - 0 4 [2, 1, 2, 2, 4, 5, 10, 6, 8]
  - 6 8 [1, 2, 2, 2, 4, 5, 6, 8, 10]

Selection Problem

- Given n elements from a totally ordered universe, find the k^{th} smallest
- Minimum: $k=1$, maximum: $k=n$
- Median: $k=\lfloor(n+1)/2\rfloor$
- $O(n)$ comparisons for min or max
- $O(n \log n)$ comparisons by sorting
- $O(n \log k)$ comparisons with a **binary heap**

Selection Problem

- The input may or may not be ordered
  - It is orderable since drawn from an ordered universe
- Also named as i^{th} order statistic
- Applications of Selection
  - Pivot selection by median can be used for Quicksort
  - Incremental sorting
  - Order statistics

Recall Binary Heap

- A binary heap is a **complete** binary tree which satisfies the heap ordering property
  - min-heap: the value of each node is greater than or equal to the value of its parent, minimum-value element at the root
  - max-heap: the value of each node is less than or equal to the value of its parent, maximum-value element at the root
- Insertion O(log n)
- Extraction O(log n)
- Application: priority queue

Quickselect

- Idea: 3-way partition such that
  - smaller, equal and larger items in L, M and R, respectively
  - then recur in one of the subarrays - the one containing the kth smallest element
- Pick pivot $p \in A$ uniformly at random

Quickselect

```
def quickselect(A,k):
  p = A[int(len(A)/2)]  # Pick the middle
  L,M,R = partition3way(A,p)  # (n-1) comparisons
  if k<=len(L):
    return quickselect(L,k)
  elif k>len(L)+len(M):
    return quickselect(R,k-len(L)-len(M))
  else:
    return p
def partition3way(A,p):
  L,M,R=[],[],[]
  for a in A:  # Can be improved like quicksort
    if p>a: L.append(a)
    elif p<a: R.append(a)
    else: M.append(a)
  return L,M,R
```

Analysis of Quickselect

- Without losing generality, assume distinct elements
- Candy bar split problem
  - After a "fair" split the average length of the longest bar is 3/4
  - Expected value E(x) = integral 1/2 to 1 with uniform distribution
  - $E(x) = \int_{\frac{1}{2}}^{1} \mathrm{x} f(x) dx \,, f(x) = \begin{cases} 0, else \\ 2, \frac{1}{2} \leq x \leq 1 \end{cases}$
  - E(x) = 3/4
- T(n) = T(3n/4) + O(n)

Analysis of Quickselect

- Normal case: $T(n) = T(3n/4) + O(n)$
  - $T(n) \leq 4n$
  - Linear algorithm (!)
- Unlucky case: $T(n) = T(n-1) + O(n)$
  - $T(n) \leq n^2$
  - More costly than $O(n \log n)$ - sorting

Deterministic Selection

- Median of medians

1. Group the array into n/5 groups of size 5 and find the median of each group
2. Recursively, find the true median of the medians, call this p
3. Use p as a pivot to split the array into subarrays L and R
4. Recurse on the appropriate subarray

Deterministic Selection Analysis

- $O(n)$ for step 1
- $T(n/5)$ for step 2
- 3/10 subarray $\leq p$, 3/10 subarray $\geq p$
- $T(n) \leq cn + T(n/5) + T(7n/10)$
- Then, $T(n) \leq cn + T(9n/10)$
- Finally, $T(n) \leq 10n$
- Thus, the cost increases for the sake of determinism

Deterministic Selection Analysis