CS612 Algorithm Design and Analysis

Lecture 15. SELECTION problem [1]

Dongbo Bu

Institute of Computing Technology
Chinese Academy of Sciences, Beijing, China

[1] The slides are made based on Randomized Algorithm by R. Motwani and P. Raghavan, and a lecture by T. Chan.

Outline

- Introduction
- Several lower bounds;
- Algorithms for selection:
  1. Deterministic $O(n)$ algorithm;
  2. A randomized divide-and-conquer $O(n)$ algorithm;
  3. A random sampling algorithm.

Note:

- The natural brute-force algorithm is already polynomial time; divide-and-conquer is serving to reduce the running time to a lower polynomial.
- Divide-and-conquer becomes more powerful when combined with randomization.
- Given a large dataset, it might be useful to randomly sample a small set first. We can perform computation on this small set using brute-force algorithm, and finally geralize observations to the whole dataset.

Introduction

INPUT:
Given a set of number $S = \{a_1, a_2, ..., a_n\}$, and a number $k \leq n$;
OUTPUT:
the k-th smallest item in general case (or the median of S as a specical case).

For example, given a set $S = \{18, 15, 27, 13, 1, 7, 25\}$, the objective is the median of S.
Note:

- A feasible strategy is to sort S first, and then report the k-th one, which takes $O(n \log n)$ time.
- It is possible to develop a faster algorithm by using divide-and-konquoror technique, say the deterministic linear algorithm ($16n$ comparisons) by Blum et al.

Deterministic algorithm

Algorithm $Select(S, k)$:

Choose an item s_i from S as a pivot;
$S^+ = \{\}$;
$S^- = \{\}$;
for $j = 1$ to n **do**
    if $s_j > s_i$ **then**
        $S^+ = S^+ \cup \{s_j\}$;
    else
        $S^- = S^- \cup \{s_j\}$;
    end if
end for
if $|S^-| = k - 1$ **then**
    return s_i;
else if $|S^-| = k - 1$ **then**
    return $Select(S^-, k)$;
else
    return $Select(S^+, k - |S^-| + 1)$;
end if

Perform iteration on ONLY one subset.

Intuition:

1. At first, an element a_i is chosen to split S into two parts $S^+ = \{a_j : a_j \geq a_i\}$, and $S^- = \{a_j : a_j < a_i\}$.
2. We can determine whether the k-th median is in S^+ or S^-.
3. Thus, we perform iteration on ONLY one subset.

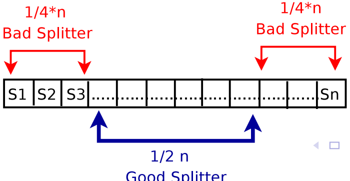
How to choose a splitter?

We have the follow options:

- Bad choice: select the smallest element at each iteration.
  $T(n) = T(n-1) + O(n) = O(n^2)$
- Ideal choice: select the median at each iteration.
  $T(n) = T(\frac{n}{2}) + O(n) = O(n)$
- Good choice: select a "centered" element a_i, i.e., $|S^+| \geq \epsilon n$, and $|S^-| \geq \epsilon n$ for a fixed $\epsilon > 0$.

$$\begin{aligned} T(n) &\leq T((1-\epsilon)n) + O(n) \\ &\leq cn + c(1-\epsilon)n + c(1-\epsilon)^2 n + ... \\ &= O(n) \end{aligned} \tag{1}$$

e.g.: $\epsilon = \frac{1}{4}$:



BFPRT algorithm: a linear deterministic algorithm I

- Still using the idea of choosing splitter. The ideal splitter is the median; however, finding the median is exactly our objective.
- Thus, just try to get "something close to the median", say within $\frac{n}{4}$ from the median.
- How can we get something close to the median? Instead of finding the median of the "whole set", find a median of a "sample".
- But how to choose a sample? Medians again!

Median of medians algorithm [Blum, 1973]

"Median of medians" algorithm:

1: Line up elements in groups of 5 elements;
2: Find the median of each group; (takes $O(\frac{6n}{5})$ time)
3: Find the median of medians (denoted as M); (takes $T(\frac{n}{5})$ time)
4: Use M as splitter to partition the input and call the algorithm recursively on one of the partitions.

One iteration on the list {0,1,2,3,...99}

	12	15	11	2	9	5	0	7	3	21	44	40	1	18	20	32	19	35	37	39
	13	16	14	8	10	26	6	33	4	27	49	46	52	25	51	34	43	56	72	79
Medians	17	23	24	28	29	30	31	36	42	47	50	55	58	60	63	65	66	67	81	83
	22	45	38	53	61	41	62	82	54	48	59	57	71	78	64	80	70	76	85	87
	96	95	94	86	89	69	68	97	73	92	74	88	99	84	75	90	77	93	98	91

Analysis:
$T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + \frac{6n}{5}$ at most $24n$ comparisons.
(here, $\frac{7n}{10}$ comes from the fact that at least $\frac{3n}{10}$ can be deleted by using M as the splitter.)

Divide-and-conquorer with random pivoting

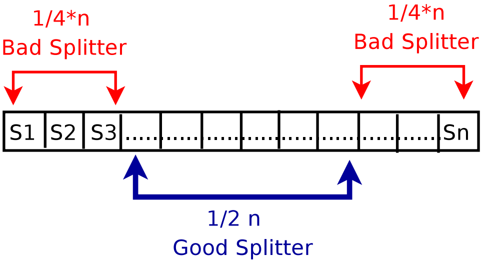
Randomized divide-and-conquorer

Algorithm $RandomSelect(n, k)$:

Choose an element s_i from S uniformly at random;
$S^+ = \{\};;$
$S^- = \{\};;$
for $j = 1$ to n **do**
    if $s_j > s_i$ **then**
        $S^+ = S^+ \cup \{s_j\};$
    else
        $S^- = S^- \cup \{s_j\};$
    end if
end for
if $|S^-| = k - 1$ **then**
    return s_i;
else if $|S^-| = k - 1$ **then**
    return $RandomSelect(S^-, k)$;
else
    return $RandomSelect(S^+, k - |S^-| + 1)$;
end if

Randomized divide-and-conquorer cont'd

e.g.: $\epsilon = \frac{1}{4}$:



Key observation: if we choose a splitter $a_i \in S$ uniformly at random, it is easy to get a good splitter since a fairly large fraction of the elements are "centered".

Randomized divide-and-conquorer cont'd

Theorem

The expected running time of Select(n,k) is $O(n)$.

Proof.

- Let $\epsilon = \frac{1}{4}$. We'll say that the algorithm is in phase j when the size of set under consideration is in $[n(\frac{3}{4})^{j-1}, n(\frac{3}{4})^j]$.
- Let X be the number of steps. And X_j be the number of steps in phase j. Thus, $X = X_0 + X_1 + ...$
- Consider the j-th phase. The probability to find a centered splitter is $\geq \frac{1}{2}$ since at least half elements are centered. Thus, the expected number of iterations to find a centered splitter is: 2.
- Each iteration costs $cn(\frac{3}{4})^j$ steps since there are at most $n(\frac{3}{4})^j$ elements in phase j. Thus, $E(X_j) \leq 2cn(\frac{3}{4})^j$.
- $E(X) = E(X_0 + X_1 + ...) \leq \sum_j 2cn(\frac{3}{4})^j \leq 8cn$.

$\square$

Random sampling strategy for selection

A "random sampling" algorithm [Floyd & Rivest, 1975]

Basic idea: randomly sample a subset as a representation of the whole set.

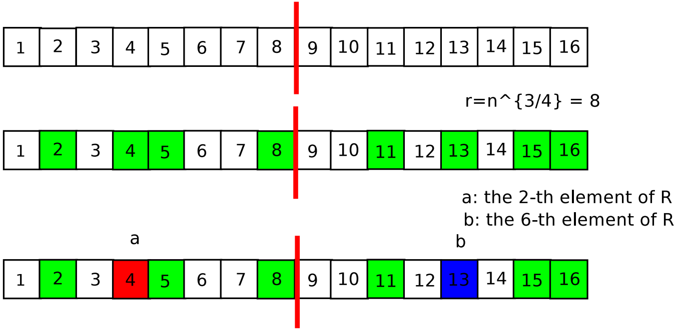
Random sampling algorithm:

1: randomly sample r elements (with replacement) from $S = \{s_1, s_2, ..., s_n\}$. Denote the r elements as R.
2: take the $(1-\delta)\frac{r}{2}$-th smallest element of R (denoted as a), and the $(1+\delta)\frac{r}{2}$-th smallest element of R (denoted as b);
3: divide S into three dis-joint subsets:

$$\begin{aligned} L &= \{s_i : s_i < a\}; \\ M &= \{s_i : a \leq s_i \leq b\}; \\ H &= \{s_i : s_i > b\}; \end{aligned}$$

4: check $|L| \leq \frac{n}{2}$, $|H| \leq \frac{n}{2}$, and $|M| \leq c\delta n$. If not, goto Step 1.
5: return the $(\frac{n}{2} - |L|)$-th smallest of M;

Example



Two requirements of M:

- On one side, M should be LARGE enough such that the median is covered by M with a high probability;
- On the other side, M should be SMALL enough such that Step 4 will not take a long time;

Time-comlexity analysis

Running time:

- Step 2: $O(r \log r) = o(n)$; (sorting R)
- Step 3: $2n$ steps. $(O(n) + O(|M| + |H|))$
- Step 4: $O(\delta n \log(\delta n))$.

Setting $r = n^{\frac{3}{4}}$, and $\delta = n^{-\frac{1}{4}}$. The time bound of Step 4 changes to:

- Step 4: $O(\delta n \log(\delta n)) = o(n)$.

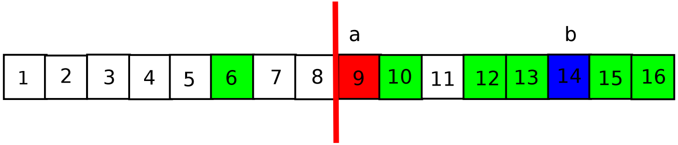
Total steps: $2n + o(n)$.

- The best known deterministic algorithm: $3n$. But too complicated.
- A lower bound: $2n$.

Error probability analysis I

Theorem

With probability $1 - O(n^{-\frac{1}{4}})$, *the* $RandomSamplingSelect$ *algorithm reports the median in the first pass. Thus, the running time is only* $2n + o(n)$.



Error probability analysis II

Three cases of failure in Step 4:
Case 1:
Define index variable $x_i = 1$ when the i-th sample is less than the median, and $x_i = 0$ otherwise. Let $X = x_1 + ... + x_r$ be the number of samples that is less than the median. We have:
$E(x_i) = \frac{1}{2}$ and $\sigma^2(x_i) = \frac{1}{4}$.
$E(X) = \frac{1}{2}r$ and $\sigma^2(X) = \frac{1}{4}r$.

$$
\begin{aligned}
\Pr(|L| \geq \tfrac{n}{2}) &= \Pr(X \leq \tfrac{1-\delta}{2}r) && (2)\\
&= \Pr(|X - E(X)| \geq \tfrac{\delta}{2}r) && (3)\\
&\leq \frac{\sigma^2(X)}{(\frac{\delta}{2}r)^2} && (4)\\
&= n^{-\frac{1}{4}} && (5)
\end{aligned}
$$

Case 2 and Case 3 are similar and thus omited.