Algorithms and datastructures I
Lecture 12: k-th smallest element; sorting

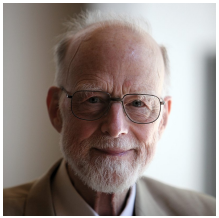
Jan Hubička

Department of Applied Mathematics
Charles University
Prague

May 5 2020

QuickSelect

Problem: Find k-th smallest element of a sequence $(x_1, \ldots, x_n)$.



Sir Tony Horae

QuickSelect($(x_1, \ldots, x_n)$, k), Sir Tony Horae 1961

1. Choose pivot p.
2. Split $(x_1, \ldots, x_n)$ to $L = \{x_i : x_i < p\}$, $E = \{x_i : x_i = p\}$, $R = \{x_i : x_i > p\}$.
3. If $k \leq |L|$: return QuickSelect(L, k).
4. If $k \leq |L| + |E|$: return x_k.
5. Return QuickSelect (L, $k - |L| - |E|$).

1. Assume that p is always maximum.
   Time complexity: $\Theta(N^2)$.
2. Assume that p is median.
   Time complexity: $T(N) = T\left(\frac{N}{2}\right) + \Theta(N) = \Theta(N)$.
3. Assume that p is "almost median" (at least $\frac{1}{4}$ of elements is smaller than p and $\frac{1}{4}$ is bigger than p).
   Time complexity: $T(N) = T\left(\frac{3N}{2}\right) + \Theta(N) = \Theta(N)$.
4. Randomized choice of almost median: Pr[random element is almost median] $\geq \frac{1}{2}$

Lemma

Let V be an event that occurs in trial with probability p.
The expected number of trials to first occurrence of V is $\frac{1}{p}$.

Expected number of trials is 2; time complexity $\Theta(N)$.

5. Random choice of pivot.
   Time complexity: $\Theta(N)$.

Median of medians algorithm

Select$((x_1, \ldots, x_n), k)$, Blum, Floyd, Pratt, Rivest, Tarjan 1973

1. If $n \leq 10$ use brute-force.
2. Divide $x_1, \ldots x_n$ to 5-tuples $P_1, \ldots P_t$, $t = \lfloor \frac{n}{5} \rfloor$.
3. For $i = 1, 2, \ldots t$: $m_i \leftarrow$Median(P_i).
4. $p \leftarrow$Select$((m_1, \ldots, m_t), \lfloor \frac{t}{2} \rfloor)$

Median of medians algorithm

Select($(x_1, \ldots, x_n), k$), Blum, Floyd, Pratt, Rivest, Tarjan 1973

1. If $n \leq 10$ use brute-force.
2. Divide $x_1, \ldots x_n$ to 5-tuples $P_1, \ldots P_t$, $t = \lfloor \frac{n}{5} \rfloor$.
3. For $i = 1, 2, \ldots t$: $m_i \leftarrow$Median(P_i).
4. $p \leftarrow$Select($(m_1, \ldots, m_t), \lfloor \frac{t}{2} \rfloor$)
5. Split $(x_1, \ldots, x_n)$ to $L = \{x_i : x_i < p\}$, $E = \{x_i : x_i = p\}$, $R = \{x_i : x_i > p\}$.
6. If $k \leq |L|$: return Select(L,k).
7. If $k \leq |L| + |E|$: return k.
8. Return Select(L, $k - |L| - |E|$).

Median of medians algorithm

$$T(n) = T(\frac{n}{5}) + T(\frac{7}{10}n) + \Theta(n)$$

Median of medians algorithm

$$T(n) = T(\frac{n}{5}) + T(\frac{7}{10}n) + \Theta(n)$$

Assume $T(n) = cn$

$$cn = \frac{1}{5}cn + \frac{7}{10}cn + \alpha n$$

Median of medians algorithm

$$T(n) = T(\frac{n}{5}) + T(\frac{7}{10}n) + \Theta(n)$$

Assume $T(n) = cn$

$$\begin{aligned} cn &= \frac{1}{5}cn + \frac{7}{10}cn + \alpha n \\ \left(1 - \frac{1}{5} - \frac{7}{10}\right) cn &= \alpha n \end{aligned}$$

Median of medians algorithm

$$T(n) = T(\frac{n}{5}) + T(\frac{7}{10}n) + \Theta(n)$$

Assume $T(n) = cn$

$$\begin{aligned} cn &= \frac{1}{5}cn + \frac{7}{10}cn + \alpha n \\ \left(1 - \frac{1}{5} - \frac{7}{10}\right) cn &= \alpha n \\ \frac{1}{10}cn &= \alpha n \end{aligned}$$

Median of medians algorithm

$$T(n) = T(\frac{n}{5}) + T(\frac{7}{10}n) + \Theta(n)$$

Assume $T(n) = cn$

$$\begin{aligned} cn &= \frac{1}{5}cn + \frac{7}{10}cn + \alpha n \\ \left(1 - \frac{1}{5} - \frac{7}{10}\right)cn &= \alpha n \\ \frac{1}{10}cn &= \alpha n \\ c &= 10\alpha \end{aligned}$$

Median of medians algorithm

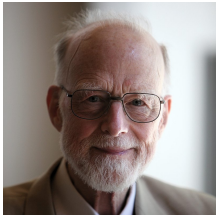
$$T(n) = T(\frac{n}{5}) + T(\frac{7}{10}n) + \Theta(n)$$

Assume $T(n) = cn$

$$\begin{aligned} cn &= \frac{1}{5}cn + \frac{7}{10}cn + \alpha n \\ \left(1 - \frac{1}{5} - \frac{7}{10}\right) cn &= \alpha n \\ \frac{1}{10}cn &= \alpha n \\ c &= 10\alpha \end{aligned}$$

Deterministic runtime: $\Theta(n)$!

QuickSort



Sir Tony Horae

QuickSort($(x_1, \ldots, x_n)$, k), Sir Tony Horae 1961

1. If $n = O(1)$: use special purpose sorting algorithm.
2. Choose pivot p.
3. Split $(x_1, \ldots, x_n)$ to $L = \{x_i : x_i < p\}$, $E = \{x_i : x_i = p\}$, $R = \{x_i : x_i > p\}$.
4. $L' \leftarrow$QuickSort (L).
5. $R' \leftarrow$QuickSort (R).
6. Return L', E, R'.

QuickSort

Sir Tony Horae

QuickSort($(x_1, \ldots, x_n)$, k), Sir Tony Horae 1961

1. If $n = O(1)$: use special purpose sorting algorithm.
2. Choose pivot p.
3. Split $(x_1, \ldots, x_n)$ to $L = \{x_i : x_i < p\}$, $E = \{x_i : x_i = p\}$, $R = \{x_i : x_i > p\}$.
4. $L' \leftarrow$QuickSort (L).
5. $R' \leftarrow$QuickSort (R).
6. Return L', E, R'.

Time complexity:

1. If p is median: $T(n) = 2T(\frac{n}{2}) + \Theta(n)$

QuickSort

Sir Tony Horae

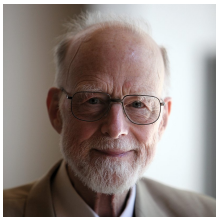
QuickSort($(x_1, \ldots, x_n)$, k), Sir Tony Horae 1961

1. If $n = O(1)$: use special purpose sorting algorithm.
2. Choose pivot p.
3. Split $(x_1, \ldots, x_n)$ to $L = \{x_i : x_i < p\}$, $E = \{x_i : x_i = p\}$, $R = \{x_i : x_i > p\}$.
4. $L' \leftarrow$QuickSort (L).
5. $R' \leftarrow$QuickSort (R).
6. Return L', E, R'.

Time complexity:

1. If p is median: $T(n) = 2T(\frac{n}{2}) + \Theta(n) = \Theta(n \log n)$

QuickSort



Sir Tony Horae

QuickSort($(x_1, \ldots, x_n)$, k), Sir Tony Horae 1961

1. If $n = O(1)$: use special purpose sorting algorithm.
2. Choose pivot p.
3. Split $(x_1, \ldots, x_n)$ to $L = \{x_i : x_i < p\}$, $E = \{x_i : x_i = p\}$, $R = \{x_i : x_i > p\}$.
4. $L' \leftarrow$QuickSort (L).
5. $R' \leftarrow$QuickSort (R).
6. Return L', E, R'.

Time complexity:

1. If p is median: $T(n) = 2T(\frac{n}{2}) + \Theta(n) = \Theta(n \log n)$
2. If p is min: $T(n) = \Theta(n^2)$

QuickSort

Sir Tony Horae

QuickSort($(x_1, \ldots, x_n)$, k), Sir Tony Horae 1961

1. If $n = O(1)$: use special purpose sorting algorithm.
2. Choose pivot p.
3. Split $(x_1, \ldots, x_n)$ to $L = \{x_i : x_i < p\}$, $E = \{x_i : x_i = p\}$, $R = \{x_i : x_i > p\}$.
4. $L' \leftarrow$QuickSort (L).
5. $R' \leftarrow$QuickSort (R).
6. Return L', E, R'.

Time complexity:

1. If p is median: $T(n) = 2T(\frac{n}{2}) + \Theta(n) = \Theta(n \log n)$
2. If p is min: $T(n) = \Theta(n^2)$
3. If p is chosen randomly: $T(n) = \Theta(n \log n)$ (proof is coming)

Random choice of pivot

Theorem

Expected number of comparisons in QuickSort with random choice of median is $\Theta(n \log n)$.

Let $y_1 < y_2 < \cdots < y_n$ be the output of the algorithm.

Random choice of pivot

Theorem

Expected number of comparisons in QuickSort with random choice of median is $\Theta(n \log n)$.

Let $y_1 < y_2 < \cdots < y_n$ be the output of the algorithm.
Denote by C_{ij} # of comparisons if x_i and x_j.

Random choice of pivot

Theorem

Expected number of comparisons in QuickSort with random choice of median is $\Theta(n \log n)$.

Let $y_1 < y_2 < \cdots < y_n$ be the output of the algorithm.
Denote by C_{ij} # of comparisons if x_i and x_j.

$$C_{ij} = \begin{cases} 0 \\ 1 \end{cases}$$

Random choice of pivot

Theorem

Expected number of comparisons in QuickSort with random choice of median is $\Theta(n \log n)$.

Let $y_1 < y_2 < \cdots < y_n$ be the output of the algorithm.
Denote by C_{ij} # of comparisons if x_i and x_j.

$$C_{ij} = \begin{cases} 0 & \\ 1 & \Pr = \frac{2}{j-i+1} \end{cases}$$

Random choice of pivot

Theorem

Expected number of comparisons in QuickSort with random choice of median is $\Theta(n \log n)$.

Let $y_1 < y_2 < \cdots < y_n$ be the output of the algorithm.
Denote by C_{ij} # of comparisons if x_i and x_j.

$$C_{ij} = \begin{cases} 0 & \\ 1 & \Pr = \frac{2}{j-i+1} \end{cases}$$

$$\mathbb{E}\left[\sum_{1 \le i < j \le n} C_{ij}\right]$$

Random choice of pivot

Theorem

Expected number of comparisons in QuickSort with random choice of median is $\Theta(n \log n)$.

Let $y_1 < y_2 < \cdots < y_n$ be the output of the algorithm.
Denote by C_{ij} # of comparisons if x_i and x_j.

$$C_{ij} = \begin{cases} 0 & \\ 1 & \Pr = \frac{2}{j-i+1} \end{cases}$$

$$\mathbb{E}\left[\sum_{1 \le i < j \le n} C_{ij}\right] = \sum_{1 \le i < j \le n} \mathbb{E}\left[C_{ij}\right]$$

Random choice of pivot

Theorem

Expected number of comparisons in QuickSort with random choice of median is $\Theta(n \log n)$.

Let $y_1 < y_2 < \cdots < y_n$ be the output of the algorithm.
Denote by C_{ij} # of comparisons if x_i and x_j.

$$C_{ij} = \begin{cases} 0 & \\ 1 & \Pr = \frac{2}{j-i+1} \end{cases}$$

$$\mathbb{E}\left[\sum_{1 \le i < j \le n} C_{ij}\right] = \sum_{1 \le i < j \le n} \mathbb{E}\left[C_{ij}\right] = \sum_{1 \le i < j \le n} \frac{2}{j-i+1}$$

Random choice of pivot

Theorem

Expected number of comparisons in QuickSort with random choice of median is $\Theta(n \log n)$.

Let $y_1 < y_2 < \cdots < y_n$ be the output of the algorithm.
Denote by C_{ij} # of comparisons if x_i and x_j.

$$C_{ij} = \begin{cases} 0 & \\ 1 & \Pr = \frac{2}{j-i+1} \end{cases}$$

$$\mathbb{E}\left[\sum_{1 \le i < j \le n} C_{ij}\right] = \sum_{1 \le i < j \le n} \mathbb{E}\left[C_{ij}\right] = \sum_{1 \le i < j \le n} \frac{2}{j-i+1} = \sum_{1 \le d \le n} \sum_{1 \le i \le n-d+1} \frac{2}{d}$$

Random choice of pivot

Theorem

Expected number of comparisons in QuickSort with random choice of median is $\Theta(n \log n)$.

Let $y_1 < y_2 < \cdots < y_n$ be the output of the algorithm.
Denote by C_{ij} # of comparisons if x_i and x_j.

$$C_{ij} = \begin{cases} 0 & \\ 1 & \Pr = \frac{2}{j-i+1} \end{cases}$$

$$\mathbb{E}\left[\sum_{1 \le i < j \le n} C_{ij}\right] = \sum_{1 \le i < j \le n} \mathbb{E}\left[C_{ij}\right] = \sum_{1 \le i < j \le n} \frac{2}{j-i+1} = \sum_{1 \le d \le n} \sum_{1 \le i \le n-d+1} \frac{2}{d}$$

$$= \sum_{1 \le d \le n} \frac{2(n-d+1)}{d} \le \sum_{1 \le d \le n} \frac{2n}{d}$$

Random choice of pivot

Theorem

Expected number of comparisons in QuickSort with random choice of median is $\Theta(n \log n)$.

Let $y_1 < y_2 < \cdots < y_n$ be the output of the algorithm.
Denote by C_{ij} # of comparisons if x_i and x_j.

$$C_{ij} = \begin{cases} 0 & \\ 1 & \Pr = \frac{2}{j-i+1} \end{cases}$$

$$\mathbb{E}\left[\sum_{1 \le i < j \le n} C_{ij}\right] = \sum_{1 \le i < j \le n} \mathbb{E}\left[C_{ij}\right] = \sum_{1 \le i < j \le n} \frac{2}{j-i+1} = \sum_{1 \le d \le n} \sum_{1 \le i \le n-d+1} \frac{2}{d}$$

$$= \sum_{1 \le d \le n} \frac{2(n-d+1)}{d} \le \sum_{1 \le d \le n} \frac{2n}{d} = \Theta(n \log n).$$

Lower bound on sorting

Definition (Comparsion model for sorting)

1. Elements being sorted can only be compared and assigned.
2. All decisions of the algorithm are deterministic

Theorem

Every sorting algorithm in the comparison model has time complexity $\Omega(n \log n)$.

Theorem

Every sorting algorithm in the comparison model has time complexity $\Omega(n \log n)$.

Theorem

Every sorting algorithm in the comparison model has time complexity $\Omega(n \log n)$.

Consider decision tree of the algorithm. Every vertex is binary and number of leaves is at least number of permutations.

Theorem

Every sorting algorithm in the comparison model has time complexity $\Omega(n \log n)$.

Consider decision tree of the algorithm. Every vertex is binary and number of leaves is at least number of permutations.

$$h \geq \log_2(n!) \geq \frac{1}{2} n \log n$$

Counting sort

We want to sort set of integer in range $\{1, 2, \ldots, r\}$ where r is relatively small.

CountingSort $((x_1, x_2, \ldots x_n), r)$

1. For $i = 1, \ldots, r$: $c_i \leftarrow 0$.
2. For $i = 1, \ldots, n$: $c_{x_i} \leftarrow c_{x_i} + 1$.
3. $j \leftarrow 1$.
4. For $i = 1, \ldots, r$:
5. Repeat c_i times: $y_j \leftarrow i, j \leftarrow j + 1$.
6. Return $y_1, \ldots, y_n$.

Counting sort

We want to sort set of integer in range $\{1, 2, \ldots, r\}$ where r is relatively small.

CountingSort $((x_1, x_2, \ldots x_n), r)$

1. For $i = 1, \ldots, r$: $c_i \leftarrow 0$.
2. For $i = 1, \ldots, n$: $c_{x_i} \leftarrow c_{x_i} + 1$.
3. $j \leftarrow 1$.
4. For $i = 1, \ldots, r$:
5. Repeat c_i times: $y_j \leftarrow i, j \leftarrow j + 1$.
6. Return $y_1, \ldots, y_n$.

Time complexity: $\Theta(n + r)$.

Bucket sort

Consider we want to sort records (k_i, d_i) where k_i is the key and d_i is some additional data.

BucketSort $(((k_1, d_1), \ldots, (k_n, d_n)), r)$

1. For $i = 1, \ldots, r$: $B_i \leftarrow$empty linked list.
2. For $i = 1, \ldots, n$: Append (k_i, d_i) to B_{k_i}
3. $L \leftarrow$ empty linked list.
4. For $i = 1, \ldots, r$: Append B_i to L.
5. Return L.

Bucket sort

Consider we want to sort records (k_i, d_i) where k_i is the key and d_i is some additional data.

BucketSort $(((k_1, d_1), \ldots, (k_n, d_n)), r)$

1. For $i = 1, \ldots, r$: $B_i \leftarrow$empty linked list.
2. For $i = 1, \ldots, n$: Append (k_i, d_i) to B_{k_i}
3. $L \leftarrow$ empty linked list.
4. For $i = 1, \ldots, r$: Append B_i to L.
5. Return L.

Time complexity: $\Theta(n + r)$.

Bucket sort

Consider we want to sort records (k_i, d_i) where k_i is the key and d_i is some additional data.

BucketSort $(((k_1, d_1), \ldots, (k_n, d_n)), r)$

1. For $i = 1, \ldots, r$: $B_i \leftarrow$empty linked list.
2. For $i = 1, \ldots, n$: Append (k_i, d_i) to B_{k_i}
3. $L \leftarrow$ empty linked list.
4. For $i = 1, \ldots, r$: Append B_i to L.
5. Return L.

Time complexity: $\Theta(n + r)$.
BucketSort is stable: relative order of elements with same keys is not changed.

Lexicographic BucketSort

LexBucketSort $((x_1, \ldots, x_n), k, r)$ where $x_i \in \{1, \ldots, r\}^k$

for every $i = 1, \ldots, n$

1. $L \leftarrow x_1, \ldots, x_n$.
2. For $i = k, k-1, \ldots, 1$:
3. Sort L using BucketSort by i-th coordinate.
4. Return L.

Lexicographic BucketSort

LexBucketSort $((x_1, \ldots, x_n), k, r)$ where $x_i \in \{1, \ldots, r\}^k$

for every $i = 1, \ldots, n$

1. $L \leftarrow x_1, \ldots, x_n$.
2. For $i = k, k-1, \ldots, 1$:
3. Sort L using BucketSort by i-th coordinate.
4. Return L.

Lemma

After l iterations the list L is sorted lexicographically using i-th to k-th coordinate.

Lexicographic BucketSort

LexBucketSort $((x_1, \ldots, x_n), k, r)$ where $x_i \in \{1, \ldots, r\}^k$

for every $i = 1, \ldots, n$

1. $L \leftarrow x_1, \ldots, x_n$.
2. For $i = k, k-1, \ldots, 1$:
3. Sort L using BucketSort by i-th coordinate.
4. Return L.

Lemma

After l iterations the list L is sorted lexicographically using i-th to k-th coordinate.

Time complexity: $\Theta(k \cdot (n + r))$. Memory: $\Theta(nk + r)$.

RadixSort

Assume that our input is $x_1, x_2, \ldots x_n \in \{1, \ldots, r\}$ where r is relatively large.
Write numbers using base b and use LexBucketSort with $k = \lfloor \log_b r \rfloor + 1$.

RadixSort

Assume that our input is $x_1, x_2, \ldots x_n \in \{1, \ldots, r\}$ where r is relatively large.
Write numbers using base b and use LexBucketSort with $k = \lfloor \log_b r \rfloor + 1$.
Time complexity: $\Theta(\log_b r \cdot (n + b)) = \Theta(\frac{\log r}{\log b} \cdot (n + b))$.

RadixSort

Assume that our input is $x_1, x_2, \ldots x_n \in \{1, \ldots, r\}$ where r is relatively large.
Write numbers using base b and use LexBucketSort with $k = \lfloor \log_b r \rfloor + 1$.

Time complexity: $\Theta(\log_b r \cdot (n + b)) = \Theta(\frac{\log r}{\log b} \cdot (n + b))$.

Putting $b = \Theta(n)$ we obtain time complexity $\Theta(\frac{\log r}{\log n} \cdot (n + b))$.

Observation

If sorted numbers are polynomially large to n, say $r \leq n^\alpha$ for some fixed α then the time complexity is linear (we get $\log r \leq \alpha \log n$).

RadixSort

Assume that our input is $x_1, x_2, \ldots x_n \in \{1, \ldots, r\}$ where r is relatively large.
Write numbers using base b and use LexBucketSort with $k = \lfloor \log_b r \rfloor + 1$.

Time complexity: $\Theta(\log_b r \cdot (n + b)) = \Theta(\frac{\log r}{\log b} \cdot (n + b))$.

Putting $b = \Theta(n)$ we obtain time complexity $\Theta(\frac{\log r}{\log n} \cdot (n + b))$.

Observation

If sorted numbers are polynomially large to n, say $r \leq n^\alpha$ for some fixed α then the time complexity is linear (we get $\log r \leq \alpha \log n$).

Excercise: adjust algorithm for sorting strings in time $\Theta(n + l)$ where l is the sum of their lengths.