Datenstrukturen und Algorithmen

Exercise 3

FS 2021

Program of today

1. Feedback of last exercise
2. Repetition theory

Throwing eggs

- What would be your strategy if you would have an arbitrary number of eggs?

Throwing eggs

- What would be your strategy if you would have an arbitrary number of eggs?
  - Binary search. Worst case: $\log_2 n$ tries.

Throwing eggs

- What would be your strategy if you would have an arbitrary number of eggs?
  - Binary search. Worst case: $\log_2 n$ tries.
- What would you do if you only had one egg?

Throwing eggs

- What would be your strategy if you would have an arbitrary number of eggs?
  - Binary search. Worst case: $\log_2 n$ tries.
- What would you do if you only had one egg?
  - Start from the bottom. n tries.

Throwing Eggs

Strategy using two eggs

- First approach: intervals of equal length: partition n into k intervals: maximum number of trials

Throwing Eggs

Strategy using two eggs

- First approach: intervals of equal length: partition n into k intervals: maximum number of trials $f(k) = k + n/k - 1$
  Minimize maximum number of trials:

Throwing Eggs

Strategy using two eggs

- First approach: intervals of equal length: partition n into k intervals: maximum number of trials $f(k) = k + n/k - 1$
  Minimize maximum number of trials:
  $f'(k) = 1 - n/k^2 = 0 \Rightarrow k = \sqrt{n}$.
  $n = 100 \Rightarrow 19$ Trials. $\Theta(\sqrt{n})$
- Second approach: take first throw trial into account by considering decreasing interval sizes. Choose smallest s such that
  $s + s - 1 + s - 2 + ... + 1 = s(s+1)/2 \geq 100 \Rightarrow s = 14$.
  Maximum number of trials: $s \in \Theta(\sqrt{n})$

Asymptotically both approaches are equally good. Practically the second way is better.

Selection algorithm

- What happens if many elements are equal?
- $99, 99, \ldots, 99$, Pivot 99, smaller partition is empty, larger $n-1$ times 99
- May degrade runtime to n^2
- Solution?

Selection algorithm

- On equality with pivot, alternate between partitions

Selection algorithm

- On equality with pivot, alternate between partitions
- Modify algorithm to return number of elements equal to pivot

2. Repetition theory

Quiz

Consider the following three sequences of snap-shots (steps) of the algorithms (a) Insertion Sort, (b) Selection Sort and (c) Bubblesort. Below each sequence provide the corresponding algorithm name.

5	4	1	3	2
1	4	5	3	2
1	2	5	3	4
1	2	3	5	4
1	2	3	4	5

5	4	1	3	2
4	1	3	2	5
1	3	2	4	5
1	2	3	4	5

5	4	1	3	2
4	5	1	3	2
1	4	5	3	2
1	3	4	5	2
1	2	3	4	5

Quiz

Consider the following three sequences of snap-shots (steps) of the algorithms (a) Insertion Sort, (b) Selection Sort and (c) Bubblesort. Below each sequence provide the corresponding algorithm name.

5	4	1	3	2
1	4	5	3	2
1	2	5	3	4
1	2	3	5	4
1	2	3	4	5

5	4	1	3	2
4	1	3	2	5
1	3	2	4	5
1	2	3	4	5

5	4	1	3	2
4	5	1	3	2
1	4	5	3	2
1	3	4	5	2
1	2	3	4	5

Quiz

Consider the following three sequences of snap-shots (steps) of the algorithms (a) Insertion Sort, (b) Selection Sort and (c) Bubblesort. Below each sequence provide the corresponding algorithm name.

5	4	1	3	2
1	4	5	3	2
1	2	5	3	4
1	2	3	5	4
1	2	3	4	5

selection

5	4	1	3	2
4	1	3	2	5
1	3	2	4	5
1	2	3	4	5

bubblesort

5	4	1	3	2
4	5	1	3	2
1	4	5	3	2
1	3	4	5	2
1	2	3	4	5

insertion

Quiz

Execute two further iterations of the algorithm Quicksort on the following array. The first element of the (sub-)array serves as the pivot.

8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	**8**	9	15	10	13

Quiz

Execute two further iterations of the algorithm Quicksort on the following array. The first element of the (sub-)array serves as the pivot.

8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	**8**	9	15	10	13

Quiz

Execute two further iterations of the algorithm Quicksort on the following array. The first element of the (sub-)array serves as the pivot.

8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	**8**	9	15	10	13
2	7	5	6	3	**8**	**9**	15	10	13
2	3	5	6	**7**	**8**	**9**	13	10	**15**

Master Method

$$T(n) = \begin{cases} aT(\frac{n}{b}) + f(n) & n > 1 \\ f(1) & n = 1 \end{cases} \qquad (a, b \in \mathbb{N}^+)$$

1. $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0 \implies T(n) = \Theta(n^{\log_b a})$
2. $f(n) = \Theta(n^{\log_b a}) \implies T(n) = \Theta(n^{\log_b a} \log n)$
3. $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n \implies T(n) = \Theta(f(n))$

Examples

Maximum Subarray / Mergesort

$$T(n) = 2T(n/2) + \Theta(n)$$

Examples

Maximum Subarray / Mergesort

$$T(n) = 2T(n/2) + \Theta(n)$$

$$a = 2, b = 2,\ f(n) = cn = cn^1 = cn^{\log_2 2} \xRightarrow{[2]} T(n) = \Theta(n \log n)$$

Examples

Naive Matrix Multiplication Divide & Conquer[1]

$$T(n) = 8T(n/2) + \Theta(n^2)$$

[1] Treated in the course later on

Examples

Naive Matrix Multiplication Divide & Conquer[1]

$$T(n) = 8T(n/2) + \Theta(n^2)$$

$$a = 8, b = 2,\ f(n) = cn^2 \in \mathcal{O}(n^{\log_2 8 - 1}) \xRightarrow{[1]} T(n) \in \Theta(n^3)$$

[1] Treated in the course later on

Examples

Strassens Matrix Multiplication Divide & Conquer[2]

$$T(n) = 7T(n/2) + \Theta(n^2)$$

[2] Treated in the course later on

Examples

Strassens Matrix Multiplication Divide & Conquer[2]

$$T(n) = 7T(n/2) + \Theta(n^2)$$

$a = 7, b = 2,\ f(n) = cn^2 \in \mathcal{O}(n^{\log_2 7 - \epsilon}) \xRightarrow{[1]}$
$T(n) \in \Theta(n^{\log_2 7}) \approx \Theta(n^{2.8})$

[2] Treated in the course later on

Examples

$$T(n) = 2T(n/4) + \Theta(n)$$

Examples

$$T(n) = 2T(n/4) + \Theta(n)$$

$a = 2, b = 4,\ f(n) = cn \in \Omega(n^{\log_4 2 + 0.5}),\ 2f(n/4) = c\frac{n}{2} \leq \frac{c}{2}n^1 \overset{[3]}{\Longrightarrow}$
$T(n) \in \Theta(n)$

Examples

$$T(n) = 2T(n/4) + \Theta(n^2)$$

Examples

$$T(n) = 2T(n/4) + \Theta(n^2)$$

$a = 2, b = 4,\ f(n) = cn^2 \in \Omega(n^{\log_4 2+1.5}),\ 2f(n/4) = \frac{n^2}{8} \leq \frac{1}{8}n^2 \xrightarrow{[3]}$
$T(n) \in \Theta(n^2)$

Algorithm NaturalMergesort(A)

Input: Array A with length $n > 0$
Output: Array A sorted

```
repeat
    r ← 0
    while r < n do
        l ← r + 1
        m ← l; while m < n and A[m + 1] ≥ A[m] do m ← m + 1
        if m < n then
            r ← m + 1; while r < n and A[r + 1] ≥ A[r] do r ← r + 1
            Merge(A, l, m, r);
        else
            r ← n
until l = 1
```

Quicksort with logarithmic memory consumption

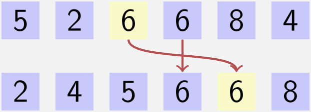
Input: Array A with length n. $1 \leq l \leq r \leq n$.
Output: Array A, sorted between l and r.

while $l < r$ **do**
- Choose pivot $p \in A[l, \ldots, r]$
- $k \leftarrow$ Partition$(A[l, \ldots, r], p)$
- **if** $k - l < r - k$ **then**
  - Quicksort$(A[l, \ldots, k-1])$
  - $l \leftarrow k + 1$
- **else**
  - Quicksort$(A[k+1, \ldots, r])$
  - $r \leftarrow k - 1$

The call of Quicksort$(A[l, \ldots, r])$ in the original algorithm has moved to iteration (tail recursion!): the if-statement became a while-statement.

Stable and in-situ sorting algorithms

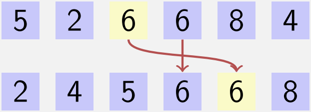
- Stable sorting algorithms don't change the relative position of two equal elements.



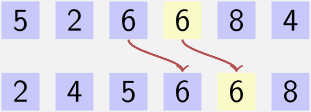
not stable

Stable and in-situ sorting algorithms

- Stable sorting algorithms don't change the relative position of two equal elements.



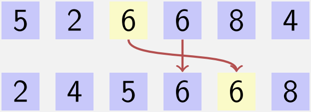
not stable



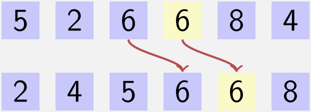
stable

Stable and in-situ sorting algorithms

- Stable sorting algorithms don't change the relative position of two equal elements.



not stable



stable

- In-situ algorithms require only a constant amount of additional memory.

  Which of the sorting algorithms are stable? Which are in-situ? (How) can we make them stable / in-situ?

Questions?