Datenstrukturen und Algorithmen

Exercise 3

FS 2018

Program of today

1. Feedback of last exercise
2. Repetition theory
3. Programming Task

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

- 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.
- What would be your strategy if you only had two 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.
- What would you do if you only had one egg?
  - Start from the bottom. n tries.
- What would be your strategy if you only had two eggs?
  - Use s tries.
  - Use decreasing interval size
  - $s + (s-1) + (s-2) + \cdots + 2 + 1 = \sum_{i=1}^{n} i = \frac{s(s+1)}{2} \geq 100$. Therefore $s = 14$.

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.
- What would be your strategy if you only had two eggs?
  - Use s tries.
  - Use decreasing interval size
  - $s + (s-1) + (s-2) + \cdots + 2 + 1 = \sum_{i=1}^{n} i = \frac{s(s+1)}{2} \geq 100$. Therefore $s = 14$.
  - $\sqrt{n}$

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
- Solutions?

Selection algorithm

- On equality with pivot, alternate between partitions

Selection algorithm

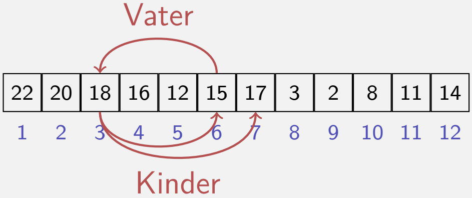
- On equality with pivot, alternate between partitions
- Modify algorithm to return number of elements equal to pivot

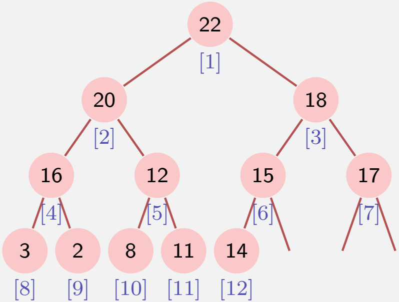
2. Repetition theory

Heap and Array

Tree $\rightarrow$ Array:

- $\text{children}(i) = \{2i, 2i + 1\}$
- $\text{parent}(i) = \lfloor i/2 \rfloor$





Depends on the starting index[1]

[1] For array that start at 0: $\{2i, 2i+1\} \rightarrow \{2i+1, 2i+2\}$, $\lfloor i/2 \rfloor \rightarrow \lfloor (i-1)/2 \rfloor$

Algorithm Sink(A, i, m)

Input : Array A with heap structure for the children of i. Last element m.
Output : Array A with heap structure for i with last element m.

```
while 2i ≤ m do
    j ← 2i; // j left child
    if j < m and A[j] < A[j + 1] then
        j ← j + 1; // j right child with greater key
    if A[i] < A[j] then
        swap(A[i], A[j])
        i ← j; // keep sinking
    else
        i ← m; // sinking finished
```

Algorithm HeapSort(A, n)

Input : Array A with length n.
Output : A sorted.

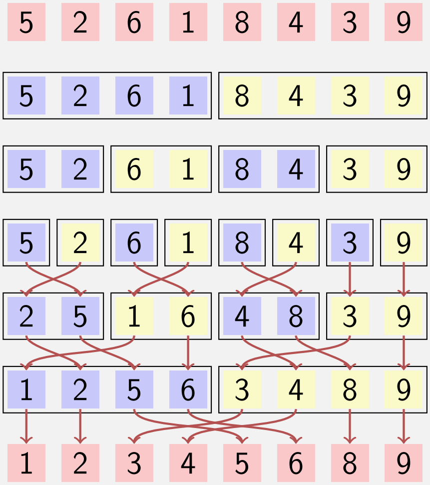
for $i \leftarrow n/2$ **downto** 1 **do**
    Sink(A, i, n);

// Now A is a heap.

for $i \leftarrow n$ **downto** 2 **do**
    swap($A[1], A[i]$)
    Sink($A, 1, i - 1$)

// Now A is sorted.

Mergesort



Split

Split

Split

Merge

Merge

Merge

Algorithm recursive 2-way Mergesort(A, l, r)

Input : Array A with length n. $1 \leq l \leq r \leq n$
Output : Array $A[l, \ldots, r]$ sorted.

if $l < r$ **then**
    $m \leftarrow \lfloor (l + r)/2 \rfloor$ // middle position
    Mergesort(A, l, m) // sort lower half
    Mergesort($A, m + 1, r$) // sort higher half
    Merge(A, l, m, r) // Merge subsequences

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 (arbitrary pivot)

2	4	5	6	8	$\boxed{3}$	7	9	1
$\boxed{2}$	1	3	6	8	$\boxed{5}$	7	9	4
1	2	3	4	5	8	$\boxed{7}$	9	6
1	2	3	4	5	6	7	9	$\boxed{8}$
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9

Algorithm Quicksort($A[l, \ldots, r]$

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

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

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 ∈ A[l, ..., r]
    k ← Partition(A[l, ..., r], p)
    if k − l < r − k then
        Quicksort(A[l, ..., k − 1])
        l ← k + 1
    else
        Quicksort(A[k + 1, ..., r])
        r ← k − 1
```

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

Stable and in-situ sorting algorithms

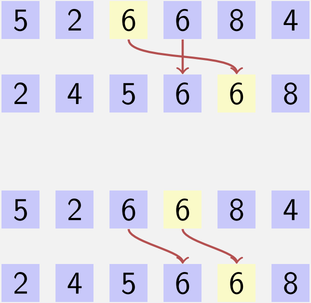
- Stabe sorting algorithms don't change the relative position of two elements.

5	2	6	6	8	4
2	4	5	6	6	8

not stable

Stable and in-situ sorting algorithms

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



not stable

stable

Stable and in-situ sorting algorithms

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

5	2	6	6	8	4
2	4	5	6	6	8

not stable

5	2	6	6	8	4
2	4	5	6	6	8

stable

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

3. Programming Task

Types as Template Parameters

```
template <typename ElementType>
class vector{
        size_t size;
        T* elem;
public:
        ...
        vector(size_t s):
        size{s},
        elem{new ElementType[s]}{}
        ...
        ElementType& operator[](size_t pos){
                return elem[pos];
        }
        ...
}
```

Function Templates

```
template <typename T> // square number
T sq(T x){
        return x*x;
}
template <typename Container, typename F>
void apply(Container& c, F f){ // x <- f(x) forall x in c
        for(auto& x: c)
        x = f(x);
}
int main(){
        std::vector<int> v={1,2,3};
        apply(v,sq<int>);
        output(v); // 1 4 9
}
```

Questions?

Questions?

Let's get to work.