Junior-Prof. Dr. Robert Elsässer, Marco Muñiz, Phillip Heidegger WS 2009/2010

Algorithms Theory, Solution for Assignment 1

http://lak.informatik.uni-freiburg.de/lak_teaching/ws09_10/algo0910.php

Exercise 1.1 - Minimum and Maximum

1.
```
function MinMaxIt(Array a) begin
  min = a[1]
  max = a[1]
  foreach elm in a do begin
    if (elm ≺ min) then min = elm
    if (max ≺ elm) then max = elm
  end
  return (min,max)
end
```

   For each element in a, $\prec$ is called two times. Hence, the number of such calls is $2 \cdot a.length = 2 \cdot n$.

$$f_{MinMaxIt}(n) = 2 \cdot n \tag{1}$$

2.
```
function min(a,b) begin
  if (a ≺ b) return a else return b
end
function max(a,b) begin
  if (a ≺ b) return b else return a
end

function MinMax(a) begin
  if (a.length > 2) begin
    mid = a.length / 2
    al, ar = split(a,mid)
    minl, maxl = MinMax(al)
    minr, maxr = MinMax(ar)
    return (min(minl,minr), max(maxl,maxr))
  end else begin
    l = 0
    r = a.length-1
    if (a[l] ≺ a[r]) then
      return (a[l],a[r])
    else
      return (a[r],a[l])
  end
end
```

   The runtime of the algorithm is:

$$f_{MinMax}(n) = \begin{cases} 2 \cdot f_{MinMax}(n/2) + 2 & \text{if } n = 2^i, i \geq 2 \\ 1 & \text{if } n = 2^i, i = 1 \end{cases}$$

   We prove $F(n) = \frac{3}{2}n - 2 = f_{MinMax}(n)$ by induction over i.

base case $i = 1, n = 2$, hence $f_{MinMax}(2) = 1$.

$$F(2) = \frac{3}{2} \cdot 2 - 2 = 1$$

induction step $i \to i+1, n \to 2 \cdot n$. Our induction hypothesis states $f_{MinMax}(2^i) = F(2^i)$. We prove $f_{MinMax}(2^{i+1}) = F(2^{i+1})$.

$$\begin{aligned} f_{MinMax}(2^{i+1}) &= f_{MinMax}(2 \cdot n) \\ &= 2 \cdot f_{MinMax}(n) + 2 \\ &=^{\text{I.H.}} 2 \cdot F(n) + 2 \\ &= 2 \cdot \left(\frac{3}{2} \cdot n - 2\right) + 2 \\ &= \frac{3}{2} \cdot 2n - 4 + 2 \\ &= F(2 \cdot n) = F(2^{i+1}) \end{aligned}$$

3. The first algorithm has runtime $f'_{MinMaxIt}(n) = 3 \cdot n$, because in every iteration a comparison is needed to break the loop if the end of the array is reached.

   The runtime for the divide-and-conquer algorithm is:

$$f'_{MinMax}(n) = \begin{cases} 2 \cdot f'_{MinMax}(n/2) + 3 & \text{if } n = 2^i, i \geq 2 \\ 2 & \text{if } n = 2^i, i = 2 \end{cases}$$

   We prove that $F'(n) = \frac{5}{2}n - 3 = f'_{MinMax}(n)$ for $n = 2^i, i \geq 1$ by induction over i.

   base case $i = 1, n = 2$, hence $f'_{MinMax}(n) = 2 = F'(2)$.

   induction step $i \to i+1, n \to 2 \cdot n$ It holds $f'_{MinMax}(n) = F'(n)$, we show that $f'_{MinMax}(2n) = F'(2n)$.

$$\begin{aligned} f'_{MinMax}(2n) &= 2 \cdot f'_{MinMax}(n) + 3 \\ &=^{\text{I.H.}} 2 \cdot F'(n) + 3 \\ &= 2 \cdot \left(\frac{5}{2}n - 3\right) + 3 \\ &= \frac{5}{2} \cdot 2n - 3 \\ &= F'(2n) \end{aligned}$$

Exercise 1.2 - Closest Pair

- Sort $(p_3, p_8, p_4, p_5, p_7, p_2, p_6, p_1)$ according to their x-coordinates in non-decreasing order.
- mindist(S):
  $S_l = \{p_3, p_8, p_4, p_5\}$
  $S_r = \{p_7, p_2, p_6, p_1\}$
  $d_l = mindist(S_l) = \sqrt{10}$
  $d_r = mindist(S_r) = \sqrt{10}$
  distance bound $d = min(d_l, d_r) = \sqrt{10}$
  Points within bound: (p_4, p_5, p_7)
  Points with minimal distance:(p_4, p_7) distance: $\sqrt{5}$

- mindist(S_l):
  $S_{ll} = \{p_3, p_8\}$
  $S_{lr} = \{p_4, p_5\}$
  $d_{ll} = mindist(S_{ll}) = \sqrt{10}$
  $d_{lr} = mindist(S_{lr}) = \sqrt{101}$
  distance bound $d = min(d_{ll}, d_{lr}) = \sqrt{10}$
  x Points within bound: (p_4, p_8), distance: $\sqrt{180}$

- mindist(S_r):
  $S_{rl} = \{p_7, p_2\}$
  $S_{rr} = \{p_6, p_1\}$
  $d_{rl} = mindist(S_{rl}) = \sqrt{10}$
  $d_{rr} = mindist(S_{rr}) = \sqrt{225}$
  distance bound $d = min(d_{rl}, d_{rr}) = \sqrt{10}$
  Points within bound: (p_2, p_6), distance: $\sqrt{50}$

- mindist(S_{ll}) (Base case):
  Points with minimal distance:(p_3, p_8), distance: $\sqrt{10}$

- mindist(S_{lr}) (Base case):
  Points with minimal distance:(p_4, p_5), distance: $\sqrt{101}$

- mindist(S_{rl}) (Base case):
  Points with minimal distance:(p_7, p_2), distance: $\sqrt{10}$

- mindist(S_{rr}) (Base case):
  Points with minimal distance:(p_6, p_1), distance: $\sqrt{153}$

Exercise 1.3 - Geometric Divide-and-Conquer

1. Evaluate for sets S_1, S_2
   $\mathsf{ReportCuts}(S_1) = \{(A, c), (B, b), (B, c)\}$
   $\mathsf{ReportCuts}(S_2) = \{(C, f), (D, f), (D, g), (E, e)\}$

2. $L(S_1) = \{A\}$
   $L(S_2) = \{D\}$

   $R(S_1) = \emptyset$
   $R(S_2) = \{C, E\}$

   $V(S_1) = \{a, b, c, d\}$
   $V(S_2) = \{e, f, g, h\}$

   $L(S) = L(S_1) \setminus R(S_2) \cup L(S_2) = \{A, D\}$
   $R(S) = R(S_2) \setminus L(S_1) \cup R(S_1) = \{C, E\}$

3. Merge step: $M = M_1 \cup M_2$
   We use a predicate $Cut(P, p)$ to express that the y value of P is between the y values of p (i.d. $p.y \leq P.y \leq p.y$). then:

   $M_1 = \{(P, p) \mid P \in R(S_2) \setminus L(S_1) \wedge p \in V(S_1) \wedge Cut(P, p)\} = \{(C, b), (C, d), (E, a)\}$
   $M_2 = \{(P, p) \mid P \in L(S_1) \setminus R(S_2) \wedge p \in V(S_2) \wedge Cut(P, p)\} = \{(A, h)\}$

   Finally:

   $\mathsf{ReportCuts}(S) = \mathsf{ReportCuts}(S_1) \cup \mathsf{ReportCuts}(S_2) \cup M$

Exercise 1.4 - Closest Pair - Memory

Let M be the amount of memory the function NächstesPaarRec needs for n points. The function NächstesPaarRec calls Merge, which itself needs linear space. Hence we can assume $M \in \Omega(n)$.

To show $M \in \Theta(n)$ it is sufficient to show $M \in \mathcal{O}(n)$.

An upper bound for the memory requirement of NächstesPaarRec is:

$$S(n) = \begin{cases} S(\frac{n}{2}) + c_1 n & \text{if } n > 2 \\ c_2 & \text{if } n \leq 2 \end{cases}$$

The base case, $n \leq 2$, needs constant space c_2, which is trivial.

In the case $n > 2$ the function has two recursive calls. An upper bound for each recursive call is $S(n/2)$. Because the two calls happens after each other, only one of these factors $S(n/2)$ is needed in our upper bound for $S(n)$.

The function calls the Merge function, which needs linear space $(c_1' n)$. The function NächstesPaarRec itself needs some constant space c_1''. We define $c_1 := c_1' + c_1''$. Hence it holds:

$$c_1' n + c_1'' \leq c_1 n \quad .$$

Thus, $S(n) \geq M(n)$ for all $n \in \mathbb{N}$.

We show $S \in \mathcal{O}(n)$, which will imply $M \in \mathcal{O}(n)$. To show $S \in \mathcal{O}(n)$ it is sufficient to show:

$$\exists c : \forall n \in \mathbb{N} : S(n) \leq cn$$

The hint in the exercise allows us to prove the above equation for $n \in \{2^i \mid i \in \mathbb{N}, i \geq 1\}$ only.

We choose $c = 2 \cdot (c_1 + c_2)$ and proceed by induction over i.

base case $i = 1, n = 2$

$$S(n) = c_2 \leq 2 \cdot (c_1 + c_2) = c \leq cn$$

induction step $i \to i + 1$, $n \to 2n$

$$\begin{aligned} S(2^{i+1}) &= S(2n) \\ &= S(n) + 2c_1 n \\ &\leq^{\text{I.H.}} cn + 2c_1 n \\ &= 2(c_1 + c_2)n + 2c_1 n \\ &= 2n(c_1 + c_2 + c_1) \\ &\leq 2n(2 \cdot (c_1 + c_2)) = 2c \cdot n \end{aligned}$$