9. domácí úkol | Vilém Zouhar

1

- $log_b(a) = log_2(4) = 2; f(n) \in O(n^{2-\epsilon}) \Rightarrow T(n) \in \Theta(n^2)$
- $log_b(a) = log_3(3) = 1; f(n) \in \Theta(n^1 \log^0(n)) \Rightarrow T(n) \in \Theta(n \cdot \log(n))$
- $log_b(a) = log_4(16) = 2; f(n) \in \Sigma(n^{2+\epsilon}) \Rightarrow T(n) \in \Theta(n!)$

2

2.1 Popis

Budeme různě skákat po poli a využívat toho, že Mirek změnil odebráním prvku paritu. Pokud se podívám na jakékoliv sudé místo v poli, tak stojím na začátku dvojice právě tehdy, pokud v dosavadní posloupnosti Mirek ještě nic neukradl. V tomto případě nás zajímá problém jen v poli $a[index,]$, opačně v $a[,index]$

2.2 Korektnost

Rozebrána v popisu.

2.3 Pseudokód

```
def solve(i, j):
        index = (j-i)/2

        if s[index-1] != s[index] and s[index-1] != s[index+1]:
                output(s[index])
                exit()

        if index % 2 == 0:
                s[index] == s[index+1]? solve(index+2, j): solve(i, index-2)
        else:
                solve(i, index)
```

2.4 Složitost

Paměťove se můžeme zanořit nejvíce v logaritmu a stejně by to šlo přepsat do iterativní formy bez zásobníku. V každém kroku dělíme problém na jeden poloviční velikosti a zbytková funkce je konstantní. To odpovídá $\Theta(n^{\log_2(1)} \cdot \log^0(n)$, tedy celkově $O(\log(n))$.

3

3.1 Popis

Využijeme pozorování, že největší obdelník vždy obsahuje alespoň jeden sloupeček. Budeme tedy procházet všechny sloupečky zleva doprava a budeme si rozmýšlet všechny jejich varianty. Začátky obdelníků budou v zásobníku a jejich pravé konce budeme procházet. Šířku obdelníku umíme spočítat v konstantě. Když načteme další sloupeček, tak budeme odebírat ze zásobníku, dokud vršek zásobníku není menší nebo roven. Při odebírání také budeme vždy kontrolovat, zdali jsme nenašli něco lepšího.

3.2 Korektnost

Opomíjíme pouze obdelníčky, které jsou obsahově menší, ostatní procházíme všechny.

3.3 Pseudokód

```
b = 0
for i in range(n):
        while gram[stack.top()] > gram[i]:
                b = best(b, stack.top()*(i-stack.pop()))
        if gram[i] > gram[i-1]:
                stack.push(i)
output(b)
```

3.4 Složitost

Každý obdelník zpracujeme právě jednou, tedy $\Theta(n)$.