Zadanie 3. (1pkt) Themis

Zadanie 4. (1pkt)

Udowodnij, że algorytm mnożenia liczb po rosyjsku jest poprawny.
Jaka jest jego złożoność czasowa i pamięciowa przy:
- jednorodnym kryterium kosztów,
- logarytmicznym kryterium kosztów?

Mamy 2 liczby a,b. W każdym kroku dzielimy (całkowicie) a przez 2 oraz mnożymy b przez 2. To oznacza, że w i-tym kroku zachodzi $b_i = b2^i$.
Jeśli w i-tym kroku a jest nieparzysta (czyli w zapisie binarnym kończy się na 1), to do wyniku dodajemy b_i, zatem algorytm daje nam następujący wynik:

$$P = \sum_{i=0}^{log_2 a} a_i b2^i = b \sum_{i=0}^{log_2 a} a_i 2^i = ba$$

Jak widać, jeśli dla danego i, $a_i = 0$, to suma nie zwiększa się, w p.p. suma zwiększa się o b_i. Po wyciągnięciu b przed sumę zauważamy, że suma ta jest równa liczbie a, stąd algorytm rzeczywiście wylicza iloczyn ab.

Kryterium jednorodne

Złożoność czasowa to $O(log_2 a)$, bo mamy pętlę, która wywoła się $log_2 a$ razy, a w każdym wywołaniu wykona operacje w czasie stałym.
Złożoność pamięciowa to O(1), bo tworzymy tylko 1 stałą P do zapamiętania sumy.

Kryterium logarytmiczne

Mamy $log_2 a$ pętli, w których dodawanie kosztuje $log_2 ab$, stąd złożoność czasowa to $O(log_2 a log_2 ab)$.
Złożoność pamięciowa to $O(log_2 ab)$, równa ilości bitów do zapamiętania w sumie.

Zadanie 5. (1pkt)

Oszacuj z dokładnością do Θ złożoność poniższego fragmentu programu:

```
res ← 0
for i ← 1 to n do
        j ← i
        while (j jest parzyste) j ← j/2
        res ← res + j
```

Łatwo zauważyć, że duży wpływ na złożoność programu ma ilość powtórzeń pętli while. W optymistycznym przypadku, gdy j jest nieparzyste, warunek pętli zostanie sprawdzony tylko raz. W pesymistycznym przypadku, gdy j jest potęgą dwójki, warunek pętli zostanie sprawdzony $\log_2 j + 1$ razy.

Zatem łączna złożoność to:

$$\left\lfloor\frac{n}{2}\right\rfloor + \left\lfloor\frac{n}{4}\right\rfloor + \cdots + \left\lfloor\frac{n}{2^{\log_2 n}}\right\rfloor = \frac{n}{2} * \frac{1-\left(\frac{1}{2}\right)^{\log_2 n}}{1-\frac{1}{2}} = n\left(1-\left(\frac{1}{2}\right)^{\log_2 n}\right) =$$

$$= n\left(1-2^{-\log_2 n}\right) = n\left(1-2^{\log_2 n^{-1}}\right) = n\left(1-\frac{1}{n}\right) = n-1$$

Zadanie 7. (1pkt)

Rozważ poniższy algorytm, który dla danego (wielo)zbioru A liczb całkowitych wylicza pewną wartość. Twoim zadaniem jest napisanie programu (w pseudokodzie), możliwie najoszczędniejszego pamięciowo, który wylicza tą samą wartość.

```
while |A| > 1 do
        a ← losowy element z A;
        A ← A \ {a}
        b ← losowy element z A;
        A ← A \ {b}
        A ← A ∪ {a – b}
 output (x mod 2), gdzie x jest elementem ze zbioru A
```

Program oblicza parzystość sumy elementów zbioru A.

Zatem możemy początkowo ustalić wartość zwracaną na 0, a następnie w każdym kroku wykonywać A[i] XOR out:

```
procedure odd_even(A[1..n])
   out<-0
   for i<-1 to n do
      out<-A[i]%2^out
   return out
```

Wtedy potrzebujemy 1 dodatkowy bit na zapamiętanie wyniku
(plus pamięć potrzebną na tablicę).

Zadanie 8. (1 pkt)

8. (1pkt) Ułóż algorytm, który dla drzewa $T = (V, E)$ oraz listy par wierzchołków $\{v_i, u_i\}$ $(i = 1, \ldots, m)$, sprawdza, czy v_i leży na ścieżce z u_i do korzenia. Przyjmij, ze drzewo zadane jest jako lista $n-1$ krawędzi (p_i, a_i), takich, że p_i jest ojcem a_i w drzewie.

```
def DFS(u)
   czas wejscia[u] <- czas
   dla v w dzieci(u)
      czas++
      DFS(v)
   czas wyjscia[u] <- czas

DFS(korzen)
jezeli czas wejscia[u] >= czas wejscia[v]
i czas wyjscia[u] <= czas wyjscia[v]
   TAK
w p. p.
   NIE
```