Zad.3

```
Procedure bubble(T[1..n])
  for i ← 1 to n-1 do
    for j ← 1 to n-1 do
      if T[j] > T[j+1] then
        SWAP(T[j], T[j+1])
```

Złożoność: Instrukcje pętli wewnętrznej wykonują się $\Theta(n^2)$, (if oraz swap sprowadzają się do stałej liczby instrukcji)

- Rozkład danych. Zawsze wykonujemy $\Theta(n^2)$ porównań, więc rozkład danych nie ma znaczenia.
- Wielkość rekordów. W najgorszym przypadku (posortowana przeciwnie) wykonamy $\Omega(n^2)$ przestawień, select zawsze wykonuje $O(n)$.
- Stabilność. Bubble sort jest stabilny (ostra nierówność gwarantuje nam zachowanie kolejności tych samych elementów
- Optymalizacja. Można dodać flagę, która monitoruje czy nie posortowaliśmy już tablicy.

zad. 4

- Zauważmy, że w każdej iteracji gdy dzielimy liczbę a przez dwa to tak naprawdę przesuwamy ją a binarnie w lewo o 1.
- Jeżeli w i-tej iteracji obecny skrajny bit a jest równy 1 (liczba a jest nieparzysta) to dodajemy do wyniku $2^{i} \cdot b$ (w i-tej iteracji liczbę b przemnożyliśmy już i-razy przez 2)

Wyliczona wartość po przejściu algorytmu:

$$\sum_{i=0}^{\log_2 a} a_i \cdot 2^i \cdot b$$

gdzie a_i to i-ty bit od lewej (skrajny w danej iteracji)

$$\parallel$$

$$b \sum_{i=0}^{\log_2 a} a_i \cdot 2^i = b \cdot a$$

- Jednorodne kryterium kosztów: W każdej iteracji wykonujemy tylko operacje arytmetyczne w czasie stałym $O(1)$, stąd złożoność czasowa wynosi $O(\log_2 a)$.

- Logarytmiczne kryterium kosztów: Pod koniec algorytmu mamy dodanie liczby rozmiaru $O(\log_2 ab)$, stąd złożoność czasowa $O(\log_2 a \cdot \log_2 ab)$

Zad 5

$$
\begin{bmatrix}
0 & 1 & 0 & \cdots & 0 & & & & \\
0 & 0 & 1 & & \vdots & & & & \\
\vdots & & \ddots & \ddots & & & & & \\
 & & & \ddots & 1 & & & & \\
0 & \cdots & \cdots & & 0 & & & & \\
m_0 & \cdots & & & m_{n-1} & a_0 & \cdots & & a_k \\
\vdots & & & & & 1 & & & \\
\vdots & & & & & 1 & 1 & & \\
\vdots & & & & & 1 & 2 & 1 & \\
\vdots & & & & & \vdots & & & \\
0 & \cdots & & & & \sum_{k=0}^{n} \binom{n}{k} & & &
\end{bmatrix}
\cdot
\begin{bmatrix}
f_i \\ f_{i+1} \\ \vdots \\ f_{i+m-1} \\ n^0 \\ n^1 \\ \vdots \\ n^k
\end{bmatrix}
=
\begin{bmatrix}
f_{i+1} \\ \vdots \\ f_{i+m} \\ (n+1)^0 \\ (n+1)^1 \\ \vdots \\ (n+1)^k
\end{bmatrix}
$$

a) Kombinacja liniowa

b) Obliczamy kolejne potęgi n z ~~wzorów skróconego~~ dwumianu newtona:

np. $(x+1)^3 = x^3 + 3x^2 + 3x + 1 = 1 + 3x + 3x^2 + x^3$

co możemy zapisać wektorowo: $\begin{bmatrix} 1 & 3 & 3 & 1 \end{bmatrix} \cdot \begin{bmatrix} x^0 \\ x^1 \\ x^2 \\ x^3 \end{bmatrix} = (x+1)^3$

np.

$$
\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1+2 \\ 1+4+4 \end{bmatrix} = \begin{bmatrix} 3^0 \\ 3^1 \\ 3^2 \end{bmatrix}
$$

zad. 6

- Wyliczona wartość zależy tylko od ostatniego bitu.

- Zauważmy, że z dokładnością do ostatniego bitu, operacja na dwóch elementach, $a, b \in A$ wewnątrz pętli jest równoważna xorowi:

$$(a-b) \bmod 2 = (a \underset{XOR}{\oplus} b) \,\&\, 1$$

$a_{\bmod 2}$	$b_{\bmod 2}$	$(a-b)_{\bmod 2}$	$a \oplus b$
0	0	0	0
0	1	1	1
1	1	0	0
1	0	1	1

- Wyliczona wartość będzie postaci: $(((\ldots a \oplus b)) \ldots \oplus \ldots c))$ itd. Zmienia tylko nawiasy, $\oplus$, i elementy A pomiędzy, bo każda iteracja, ani nie usuwa, ani nie dodaje elementów (na wymianę).

- XOR jest łączny i przemienny więc niezależnie od losowań (i otrzymanych wyników) wartość po wyjściu z pętli będzie równa: $a \oplus b \oplus c \ldots \oplus z$, $\{a, b \ldots z\} = A$

PSEUDOKOD

```
wynik ← 0
for i ← 1 to n do
    wynik ← wynik xor A[i]
return wynik mod 2
```

Zad.7

- Preprocessing: chcemy stworzyć dwie tablice pomocnicze, trzymające czas „wejścia" i „wyjścia" w kolejności przechodzenia DFS.

```
inTime ← [0] * n
OutTime ← [0] * n
time ← 1

proc DFS(v)
    inTime[v] = time++
    for u in G[v]
        if inTime[u] then
            DFS(u)
    outTime[v] = time++
DFS(0) ← korzeń
```

→ przerabiamy listę krawędzi na wygodniejszą strukturę, listę list z; G[i] - lista synów i

- Teraz możemy odpowiadać na zapytania w czasie stałym:

v jest na ścieżce do korzenia z u wtw:

$$inTime[v] < inTime[u] < outTime[v]$$

bo ten warunek jest równoważny, że u jest w poddrzewie v.

NP.