czy ostatni bit jest zapalony

$11 \to 8+2+1$

$11 \cdot 6 = 8 \cdot 6 + 2 \cdot 6 + 1 \cdot 6$

3. bubblesort $(A[1 \dots n])$

```
for i=1 to n-1:
    for j=1 to n-i:
        if (A[j] > A[j+1])
            tmp = A[j]
            A[j] = A[j+1]
            A[j+1] = tmp
```

Idea: W j-tej iteracji wewnętrznej pętli porównujemy dwie sąsiednie liczby i w razie konieczności je zamieniamy. Po ostatniej iteracji na pozycji $n-i+1$ znajduje się i-ta największa liczba.

Złożoność czasowa: $O(n^2)$, wykonuje instrukcji if posortowana lista $\to O(n)$, jeśli warunek $\to$ zapytamy ile zamian zrobił zajmuje czas stały. Zewnętrzna pętla wykona się $O(n)$ razy a w każdej jej iteracji wykona się wewnętrzna pętla, $n-i$ razy. Kod wykona się $(n-1)+(n-2)+\dots+2+1$ razy, co się równa $(n-1) \cdot n \cdot \frac{1}{2} = O(n^2)$

Złoż. pamięciowa: $O(1)$ - korzystamy jedynie ze zmiennej tmp.

Jeśli porównamy go do insert sort, będzie wypadał gorzej. Insert ma te same złożoności, ale mniej zamian, jest dwa razy szybszy od niego, tak samo szybszym wyborem będzie select sort

Również spośród tych 3, jedynie select sort jest niestabilny

6. Algorytm wylicza parzystości sumy elementów

bool wyciągamy nieparzystą to zmieniasz
wpp zostawiasz
output → parzystość

{1, 2, 3} → 1+2+3 = 6 → 0

```
bool check = 0;
for (int x : A)
    if (x % 2 == 1)
        check = !check;
return check;
```

```
for (int x : A)
    if (x % 2 == 1)
        check = !check;
return check;
```

Idea - otrzymujemy pary wierzchołków (ojciec, syn)
idziemy z wierzchołka v_i do korzenia,
jeśli jest po drodze wierzchołek v_i to znaczy
że jest on na ścieżce z korzenia do v_i
Będzie on działał w czasie $O(m \log n)$, gdzie
m - ilość par, $\log n$ - wysokość drzewa.

(1,1)

bool = 0, jeśli $v_i = v_i$ to bool = 1
weź z listy parę (p_i, u_i)

dopóki $p_i \neq$ korzeń
jeśli $p_i = v_i$ to bool = 1 zakończ

wpp
weź parę (x_i, p_i);
$p_i = x_i$
zakończ

4. Mamy podane liczby a i b, w każdym kroku liczbę a dzielimy przez 2 a liczbę b mnożymy przez 2. Wykonujemy aż nasze $a \neq 1$. Jeśli w kroku a jest niepanysta, to do całej sumy dodajemy b w obecnym kroku. Suma końcowa to wynik mnożenia

Jeśli nasze a przedstawimy binarnie, to w każdym kroku dzielimy przez 2, czyli przesuwamy binarnie w lewo - usuwamy ostatni bit. Jeśli zapis a kończy się na 1 to do sumy dodajemy b

W i-tym kroku nasze b to tak naprawdę $2^i \cdot b$

Możemy zapisać działanie algorytmu jako:

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

Jeśli i-ty bit liczby a to zero, suma nie jest zwiększona, jeżeli 1 to zwiększamy o liczbę b w tym kroku. Możemy to zapisać jako:

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

Tu możemy zauważyć że $\sum_{i=0}^{\log_2 a} a_i 2^i$ to po prostu nasza liczba a, więc mamy $a \cdot b$

Algorytm jednorodne → w każdym przejściu pętli dzielimy, mnożymy, czasem dodajemy. Wykonujemy to dokładnie $\log_2 a$, każde działanie w czasie $O(1)$ stąd otrzymujemy $O(\log_2 a)$. Pamięciowo to $O(1)$, bo z każdym przejściem przechowujemy wolne liczby.

Alg. logarytmiczne → złożoność czasowa to $\log_2 a \cdot \log_2 b$ czyli pętle z długość b w najgorszym wypadku czyli $\log_2 b$ razy stąd $O(\log_2 a \log_2 b)$. Pamięciowo to najwięcej będą liczby b w n-tym kroku i suma końcowa. Obie mogą mieć maks. $\log_2 nm$ czyli $O(\log_2 nm)$