Algorytmy i struktury danych

Sortowanie i kopce

Przygotowanie do kolokwium

Przyjmując, że `tabA[] = {1, 2, 3, 4, 5, 6, 7}` oraz `tabB[] = {7, 6, 5, 4, 3, 2, 1}` i stosując algorytmy sortujące ściśle według procedur z pliku sort2023.cc wykonaj polecenia:

Zadanie 1

Ile dokładnie porównań (między elementami tablic) wykona `insertion_sort(tabB)`, a ile `insertion_sort(tabA)`?

Przypadek optymistyczny

Dla tablicy posortowanej rosnąco o długości n, `insertion_sort` wykona **n-1 porównań**.
`insertion_sort` dla tablicy tabA wykona **6 porównań**.
Złożoność czasowa: $\boldsymbol{O(n)}$

Przypadek pesymistyczny

Dla tablicy posortowanej malejąco o długości n, `insertion_sort` wykona $\frac{n^2-n}{2}$ **porównań**.
`insertion_sort` dla tablicy tabA wykona **21 porównań**.
Złożoność czasowa: $\boldsymbol{O(n^2)}$

Zadanie 2

Ile co najwyżej porównań (między elementami tablic) wykona procedura scalająca `merge` dwie tablice n-elementowe?

Procedura scalająca dwie tablice n-elementowe `merge` wykona co najwyżej $\mathbf{2n-1}$ **porównań**
w przypadku, gdy elementy tablic są posortowane naprzemiennie rosnąco.
Złożoność czasowa procedury merge dla każdego przypadku: $\boldsymbol{O(n)}$

```
void merge(int n, int k, double leftTable[], double rightTable[]) {
int i = 0;
int j = k;
int l = 0;
while (i < k && j < n)
    if (leftTable[i] <= leftTable[j])
        rightTable[l++] = leftTable[i++];
    else
        rightTable[l++] = leftTable[j++];

while (i < k)
    leftTable[--j] = leftTable[--k];

for (i = 0; i < j; i++)
    leftTable[i] = rightTable[i];
}
```