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 $\boldsymbol{\frac{n^2-n}{2}}$ **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 $\boldsymbol{2n-1}$ **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];
}
```

Zadanie 3

Jaka jest pesymistyczna złożoność czasowa procedury `merge_sort`? Odpowiedź uzasadnij.

Złożoność czasowa procedury `merge_sort` wynosi $\boldsymbol{O(nlogn)}$, ponieważ każde wywołanie procedury `merge_sort` dzieli tablicę na dwie części o połowie długości, a następnie wywołuje procedurę `merge` na tych dwóch częściach. Złożoność czasowa nie zależy od ilości elementów i tego jak są posortowane. Algorym `merge_sort` składa się z dwóch etapów: divide i merge.

Złożoność czasowa etapu divide wynosi $\boldsymbol{O(n)}$, ponieważ wykonujemy $\boldsymbol{n-1}$ podziałów. Złożoność czasowa etapu merge wynosi $\boldsymbol{O(nlogn)}$, ponieważ wykonujemy $\boldsymbol{lbn}$ scaleń mając zawsze do dyspozycji $\boldsymbol{n}$ elementów.

Razem złożoność czasowa algorytmu `merge_sort` wynosi: $\boldsymbol{O(n) + O(nlogn) = O(nlogn)}$

Dowód metodą rekurencji uniwersalnej:

$$\boldsymbol{T(n) = 2T(n/2) + O(n)}$$

$$\boldsymbol{f(n) = n^{log_2 2} = n}$$

Rozważamy drugi przypadek:

$$\boldsymbol{T(n) = O(n \log n)}$$

Zadanie 4

Ile co najwyżej porównań (między elementami tablicy) wykona procedura partition?

Procedura partition wykona co najwyżej $\boldsymbol{n+1}$ **porównań**.

```
int partition(double t[], int n)
{
    int k = -1;
    double x = t[n / 2];
    for (;;) {
        do
            ++k;
        while (t[k] < x);

        do
            --n;
        while (t[n] > x);

        if (k < n)
            std::swap(t[k], t[n]);
        else
            return k;
    }
}
```

Zadanie 5

Jak jest średnia a jaka pesymistyczna złożoność `quick_sort`. Odpowiedź uzasadnij.

Średnia złożoność czasowa algorytmu `quick_sort` wynosi $\boldsymbol{O(nlogn)}$.
pesymistyczna złożoność czasowa algorytmu `quick_sort` wynosi $\boldsymbol{O(n^2)}$.

```
void quick_sort(double t[], int n)
{
    if (n > 1) {
        int k = partition(t, n);  // podziel na dwie czesci
        quick_sort(t, k);         // posortuj lewa
        quick_sort(t + k, n - k); // posortuj prawa
    }
}
```