Zajęcia C-5: “Sortowanie przez scalanie”

Cel zajęć i efekty uczenia

Główne cele zajęć / materiał do opanowania:

- Algorytm sortowania przez scalanie
- Algorytm obliczania liczby inwersji w ciągu

Dodatkowe cele:

- Przypomnienie i utrwalenie idei rekursji

Zadania do rozwiązania na sprawdzarce

Sierżant
Dana jest tablica A zawierająca n liczb. Posortować ją, oraz obliczyć, ile było w niej inwersji (takich par x,y, dla których x<y, ale A[x]>A[y]).

Plan zajęć

Szacunkowy czas trwania: 2 godziny lekcyjne.

1. Scalanie
    - *Algorytm scalania jest opisany w wielu podręcznikach, ale na ogół w wersji z tablicą. **Bardzo** wygodne - i dużo łatwiejsze dydaktycznie - jest użycie w nim wektorów: piszemy funkcję scal(vector<int> &A, vector<int> &B, vector<int> &C), której zadaniem jest scalić dwa posortowane wektory w jeden.*
    - *Przy implementacji scalania warto zatrzymać się nad sytuacją, w której w jednym wektorze już skończyły się elementy. Być może najprostsza do wytłumaczenia implementacja funkcji scal(A,B,C) jest następująca:*

```
i = 0; j = 0;
dla k = 1, 2, …, A.size()+B.size():
        jeśli i < A.size() to x = A[i], inaczej x = nieskończoność
        jeśli j < B.size() to y = B[j], inaczej y = nieskończoność
        jeśli x <= y to:
                C.push_back(x)
                i = i+1
        inaczej:
                C.push_back(y)
                j = j+1
```

2. Rekursja i sortowanie
    - *Przy opisie algorytmu warto przypomnieć zasady rekursji:*

        - *wywoływanie się na mniejszych danych (tutaj: dwa razy mniejszych tablicach wektorach)*
        - *możliwość policzenia ostatecznego wyniku z wyników dla mniejszych danych (tutaj: scalanie)*
        - *znany początek rekursji (tablic długości 1 nie trzeba sortować)*
    - *Znowu, algorytm jest dużo łatwiejszy na wektorach niż tablicach: wektor wejściowy rozbijamy na dwa mniejsze, na każdym wywołujemy się rekurencyjnie, a potem scalamy z powrotem do wyjściowego. Jest to mniej efektywne niż oryginalna implementacja (wektory są dość powolne), ale znacznie prostsza do wyjaśnienia uczniom. W szczególności, funkcja jest jednoargumentowa: piszemy sortuj(A), zamiast mniej intuicyjnego sortuj(A,poczatek,koniec).*
3. Analiza algorytmu, złożoność
    - *Bardzo zalecane jest przeanalizowanie całego algorytm na wybranej tablicy (8-10 elementów), ze wszystkimi wywołaniami rekurencyjnymi i wszystkimi operacjami scalania, żeby uczniowie dokładnie przekonali się, co się dzieje w algorytmie.*
    - *Mając takie drzewo wywołań, już względnie łatwo oszacować złożoność: na każdym poziomie rekursji jest O(n) operacji - na pierwszym n operacji przy scalaniu, na drugim 2*n/2 (dwie tablice po n/2), na trzecim 4*n/4, itd. Zatem złożonosć to n*liczba poziomów rekursji, czyli n*log n.*
4. Liczenie inwersji
    - *Liczba inwersji to liczba par, w których większy element stoi przed mniejszym. Zauważamy, że ponieważ algorytm sortuje tablicę, wystarczy liczyć, ile razy mniejszy element “przeskoczy” nad większym podczas przestawiania go w tablicy. Dzieje się tak tylko podczas wywoływania instrukcji:*

        ```
        C.push_back(y)
        j = j+1
        ```

      Wtedy element y przeskakuje nad tyloma większymi elementami, ile pozostało jeszcze na wektorze A (czyli A.size()-i) - taką liczbę trzeba w tym momencie doliczyć do liczby inwersji.
    - *Trzeba przypomnieć uczniom, że liczba inwersji może być kwadratowa, a więc nie mieści się w 32-bitowym typie int.*