Wstęp do programowania

Dziel i rządź

Divide et impera

- Starożytni Rzymianie znali tę zasadę
- Łatwiej się rządzi, jeśli poddani są podzieleni
- Nie chodziło im jednak bynajmniej o podziały administracyjne
- Chodziło o to, zeby skłócić poddanych!
- ... my zrobimy to bardziej humanitarnie.

Zasada dziel i rządź w programowaniu

- Chodzi o to, żeby po pierwsze umieć rozwiązać problem dla małych danych
- ... a po drugie, żeby mając duże dane tak je podzielic, aby problem sprowadzić do kilku mniejszych podproblemów, a potem scalić wyniki.
- Idealnie nadaje się do tego mechanizm rekursji, ale można sie często bez niej obyć.

Znajdowanie minimum i maksimum w tablicy

- Jak znaleźć jednocześnie dwie wartości: największą i najmniejszą w tablicy?
- Rozwiązanie:
  - podzielmy tablicę na dwie „połówki”
  - rozwiążmy problem maksimum-minimum w każdej z nich, otrzymując pary (minl,maxl),(minp,maxp)
  - wynik, to para(min(minl,minp), max(maxl,maxp))
- Czy coś zyskaliśmy?

Analiza złożoności

- Gdybyśmy te dwie wartości znajdowali po kolei, wykonalibyśmy 2n-2 porównania.
- Oznaczmy przez T(n) liczbę porównań wykonywanych przez nasz algorytm dla n danych
  - T(1)=0
  - T(n)=2T(n/2)+2
- Rozwiązanie tego układu jest dość proste. Można przez indukcję pokazać, że dla $n=2^k$
  - T(n)=(3/2)n-2

Twierdzenie Pohla

- Problemu min-max nie da się rozwiązać taniej niż za pomocą ⌜(3/2)n⌝-2 porównań.
- Dowód (bardzo ciekawy) w książce L.Banachowski, A.Kreczmar „Elementy Analizy Algorytmów”.

Twierdzenie Pohla

- Problemu min-max nie da się rozwiązać taniej niż za pomocą ⌜(3/2)n⌝-2 porównań.
- Dowód (bardzo ciekawy) w książce L.Banachowski, A.Kreczmar „Elementy Analizy Algorytmów”.

Podział (1 ; n-1)

- Bardzo typowy schemat metody Dziel i Rządź, to podzielenie zadania rozmiaru n na dwie nierówne części: rozmiaru 1 i n-1.
- Jeśli potrafimy przejść od n-1 do n (dodając jedną daną) w fazie scalania, to schemat się bardzo upraszcza:
  - Jeśli n=1, to wykonaj działania kończące
  - Jesli n>1, to wykonaj rekurencyjnie zadanie dla n-1 danych, a potem scal wynik z wynikiem dla 1-elementowej danej

Sortowanie

- Sortowanie n elementów:
  - Jeśli jest jedna dana, to nic nie rób
  - Jeśli jest n>1 danych, to posortuj n-1 elementów i scal wynik z ostatnią daną
- Scalanie polega tu na wstawieniu do posortowanej n-1-elementowej tablicy jednego elementu tak, aby nie zaburzyć porządku.

Sortowanie przez proste wstawianie

```
procedure InsertionSort(var T:tab;k:Integer);
var j:Integer;
begin {sortujemy T[1..k]}
 if k>1 then begin
    InsertionSort(T, k-1);
    j:=k-1; v:=T[k];
    while (j>0) and (T[j]>k) do begin
        T[j+1]:=T[j];
        j:=j-1
    end;
    T[j+1]:=v
 end;
end;
```

Sortowanie przez proste wstawianie iteracyjnie

```
procedure InsertionSort(var T:tab;k:Integer);
var j,k:Integer;
begin
 for k:=2 to n do begin
   j:=k-1; v:=T[k];
   while (j>0) and T[j]>k do begin
       T[j+1]:=T[j];
       j:=j-1
   end;
   T[j+1]:=v
 end;
end;
```

Sortowanie przez proste wstawianie iteracyjnie

```
procedure InsertionSort(var T:tab;k:Integer);
var j,k:Integer;
begin
 for k:=1 to n do begin
   j:=k-1; v:=T[k];
   while (j>0) and T[j]>k do begin
       T[j+1]:=T[j];
       j:=j-1
   end;
   T[j]:=v
 end;
end;
```

Analiza kosztu

- Mamy tu prostą sytuację:
  - $T(1)=0$
  - $T(n)=T(n-1)+n-1$
- To równanie rekurencyjne daje nam rozwiązanie $T(n)=(n-1)+(n-2)+...+1$, czyli $T(n)=n(n-1)/2$

Sortowanie przez scalanie

- Jeśli jednak dokonamy podziału na „połowy”, to możemy otrzymać znacznie ciekawszy algorytm.
- Wystarczy posortować połówki, a potem je scalić (koszt scalenia liniowy)

Mergesort

```
procedure mergesort(var T:Tab;l,p:Integer);
var s:Integer;
begin
 if l<p then begin
  s:=(l+p) div 2;
  Mergesort(T,l,s);
  Mergesort(T,s+1,p);
  scal(T,l,s,p);{Scalamy T[l..s] z T[s+1..p]}
end;
```

scal

```
procedure scal(var T:Tab;l,s,p:Integer);
var k1,k2,k:Integer; A:tab;
begin
 k:=l; k1:=l; k2:=s+1;
 while (k1<=s) and (k2<=p) do begin
  if T[k1]<T[k2] then begin
    A[k]:=T[k1]; k1:=k1+1 end
  else begin
    A[k]:=T[k2]; k2:=k2+1 end;
  k:=k+1;
 for k1:=k1 to s do begin
   T[k]:=T[k1]; k1:=k1+1  end;
 for k2:=k2 to s do begin
   T[k]:=T[k1]; k1:=k1+1 end;
end;
T:=A  end;
```

Analiza kosztu Mergesort

- Liczba porównań dla danych rozmiaru n, to T(n):
  - T(1)=0
  - T(n)=2T(n/2)+n
- Rozwiązanie tego równania, to funkcja T(n)=n logn

Quicksort

- Algorytm Quicksort też jest szczególnym przypadkiem zasady Dziel i Rządź.
- Tu wybieramy jakiś element (np. A[1]) i względem niego dokonujemy podziału na elementy mniejsze lub równe jemu oraz większe (lub równe). Ten podział realizujemy np. algorytmem flagi polskiej.
- Faza rządzenia, to tylko odpowiednie wywołanie rekursji.