ALGORYTMY I STRUKTURY DANYCH
WYKŁAD III (materiały pomocnicze)

Problem sortowania

Polsko Japońska Wyższa Szkoła Technik Komputerowych

Warszawa, 10 listopada 2008

Plan wykładu:

- problem sortowania:
  - algorytm sortowania przez selekcję,
  - algorytm sortowania przez wstawianie,
  - algorytm sortowania szybkiego,
- drzewa decyzyjne i dolne ograniczenie dla problemu sortowania,

Plan wykładu c.d.:

- sortowanie przez scalanie,
  - scalanie ciągów uporządkowanych,
  - złożoność algorytmu,
- sortowanie a stabilność i „działanie w miejscu”,
- sortowanie w czasie liniowym:
  - algorytm sortowania kubełkowego.
  - algorytm sortowania pozycyjnego,
  - algorytm sortowania przez zliczanie.

Problem sortowania

(algorytm sortowania przez selekcję)

Problem sortowania – algorytm sortowania przez selekcję

Zadanie (problem sortowania). Niech A będzie tablicą n różnych liczb naturalnych, gdzie $n > 0$. Podaj algorytm, który uporządkuje elementy tablicy A w kolejności rosnącej.

Idea algorytmu sortowania przez scalanie. Niech $i = 0$,

- $n - 1$-krotnie powtórz następujące działanie:
  - wyszukaj element najmniejszy wśród elementów $A[i], A[i+1], \ldots, A[n-1]$, niech to będzie element $A[min]$,
  - zamień element $A[min]$ z elementem $A[i]$,
  - zwiększ i o jeden.

Zadanie. Przedstaw działanie algorytmu sortowania przez selekcję dla następujących danych wejściowych:

$$A = [10, 7, 6, 4, 2, 11, 16, 8, 3, 1, 9].$$

Pytanie. Jaka jest złożoność czasowa algorytmu sortowania przez selekcję w przypadku średnim?

Pytanie. Jaka jest złożoność czasowa algorytmu sortowania przez selekcję w przypadku pesymistycznym?

Pytanie. Jaka jest złożoność pamięciowa algorytmu algorytmu sortowania przez selekcję?

Problem sortowania

(algorytm sortowania przez wstawianie)

Problem sortowania – algorytm sortowania przez wstawianie

Idea algorytmu sortowania przez wstawianie. Niech $i = 1$,

- $n - 1$-krotnie powtórz następujące działanie:
  - dopóki $A[i] < A[i-1]$, zamień $A[i]$ z $A[i-1]$,
  - zwiększ i o jeden.

Zadanie. Przedstaw działanie algorytmu sortowania przez wstawianie dla następujących danych wejściowych:

$$A = [10, 7, 6, 4, 2, 11, 16, 8, 3, 1, 9].$$

Pytanie. Jaka jest złożoność czasowa algorytmu sortowania przez wstawianie w przypadku średnim?

Pytanie. Jaka jest złożoność czasowa algorytmu sortowania przez wstawianie w przypadku pesymistycznym?

Pytanie. Jaka jest złożoność pamięciowa algorytmu algorytmu sortowania przez wstawianie?

Problem sortowania – algorytm sortowania przez wstawianie

Pytanie. Jaki jest koszt algorytmu sortowania przez wstawianie dla uporządkowanej rosnąco tablicy rozmiaru 100?

Pytanie. Jaki jest koszt algorytmu sortowania przez selekcję dla uporządkowanej rosnąco tablicy rozmiaru 100?

Pytanie. Jaka jest złożoność czasowa algorytmu sortowania przez wstawianie, jeżeli za operację dominującą przyjmiemy czynność przestawiania elementów tablicy A?

Pytanie. Jaka jest złożoność czasowa algorytmu sortowania przez selekcję, jeżeli za operację dominującą przyjmiemy czynność przestawiania elementów tablicy A?

Pytanie. Jak zmodyfikować algorytm sortowania przez wstawianie tak, aby złożoność rozwiązania (mierzona liczbą operacji porównań elementów tablicy A) była rzędu $n \lg n$, gdzie n jest rozmiarem tablicy A? Czy modyfikacja ta zmienia także złożoność algorytmu względem liczby operacji przestawiania elementów?

Problem sortowania

(algorytm sortowania szybkiego)

Problem sortowania – algorytm sortowania szybkiego

Idea algorytmu sortowania szybkiego. Powtarzaj rekurencyjnie następujący schemat działania:

- wybierz dowolny element aktualnie rozważanego fragmentu tablicy A, tzw. *medianę*, niech będzie to $A[m]$,
- rozdziel elementy aktualnie rozważanego fragmentu tablicy na elementy mniejsze od $A[m]$, tzw. *część młodsza* tablicy, oraz elementy większe od $A[m]$, tzw. *część starsza* tablicy,
- umieść element $A[m]$ w tablicy A tak aby poprawnie rozdzielał część młodszą od starszej,
- posortuj rekurencyjnie młodszą część tablicy,
- posortuj rekurencyjnie starszą część tablicy.

Zadanie. Przedstaw działanie algorytmu sortowania szybkiego dla następujących danych wejściowych:

$$A = [10, 7, 6, 4, 2, 11, 16, 8, 3, 1, 9] .$$

Problem sortowania – algorytm sortowania szybkiego

Rozwiązanie. Algorytm sortowania szybkiego:

```
void QuickSort(int A[n],int l,int r) { // wp:  n > 0
   int m;

   m=Rozdziel(A,l,r);

   QuickSort(A,l,m-1);
   QuickSort(A,m+1,r);
}
```

gdzie Rozdziel to np. procedura Split albo Partition.

Problem sortowania – algorytm sortowania szybkiego

Przypadek pesymistyczny. Elementy n-elementowej tablicy A posortowane są rosnąco albo malejąco, procedura rozdzielania została zaimplementowana zgodnie z metodą Split albo Partition, wtedy:

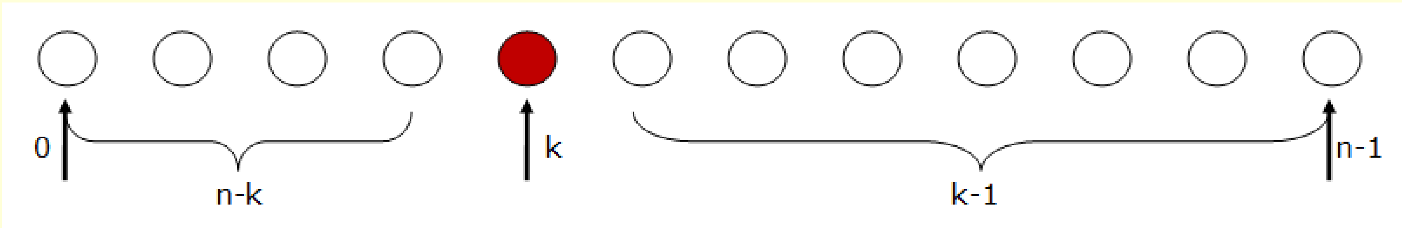
$$W(n) = \begin{cases} 0 & \text{dla } n = 1 \\ n - 1 + W(n-1) & \text{dla } n > 1 \end{cases},$$

czyli

$$\begin{aligned} W(n, 1) &= n - 1 + W(n-1) = n - 1 + n - 2 + W(n-2) = \ldots = \\ &= \ldots = n - 1 + n - 2 + \ldots + 0 = \frac{n(n-1)}{2} = \Theta\left(n^2\right). \end{aligned}$$

Problem sortowania – algorytm sortowania szybkiego

Przypadek średni. Rozkład elementów n-elementowej tablicy A jest jednorodny, mediana jest elementem k-tym co do wielkości, procedura rozdzielania została zaimplementowana zgodnie z metodą Split albo Partition, wtedy:



$$A(n) = \begin{cases} 0 & \text{dla } n = 1 \\ n - 1 + \frac{1}{n}\sum_{m=1}^{n}\left(A(n-k) + A(k)\right) & \text{dla } n > 1 \end{cases},$$

czyli

$$A(n) = O(n \lg n).$$

Pytanie. Jaka jest złożoność pamięciowa algorytmu sortowania szybkiego?

Drzewa decyzyjne

Drzewa decyzyjne i dolne ograniczenie dla problemu sortowania

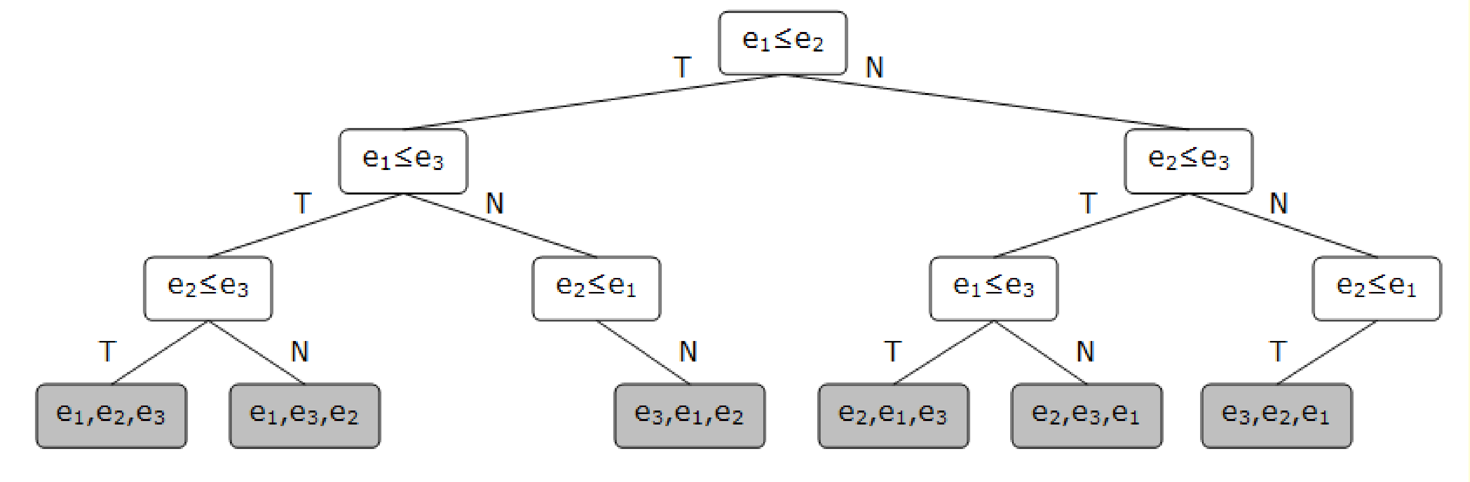
Definicja. *Drzewem decyzyjnym* algorytmu **sortowania przez porównania** nazywamy lokalnie pełne drzewo binarne (tj. każdy wierzchołek wewnętrzny drzewa ma dokładnie dwa wierzchołki następnicze), w którym:

- etykietami wierzchołków wewnętrznych są zdania opisujące relację między sortowanymi elementami,
- etykietami wierzchołków zewnętrznych (liści) są permutacje sortowanych elementów wynikające z relacji między elementami ustalonymi na podstawie etykiet wierzchołków wewnętrznych ścieżki korzeń drzewa – liść drzewa.

Wniosek. Dowolne drzewo decyzyjne algorytmu sortowania przez porównania n-elementowego ciągu wejściowego zawiera co najmniej $n!$ liści.

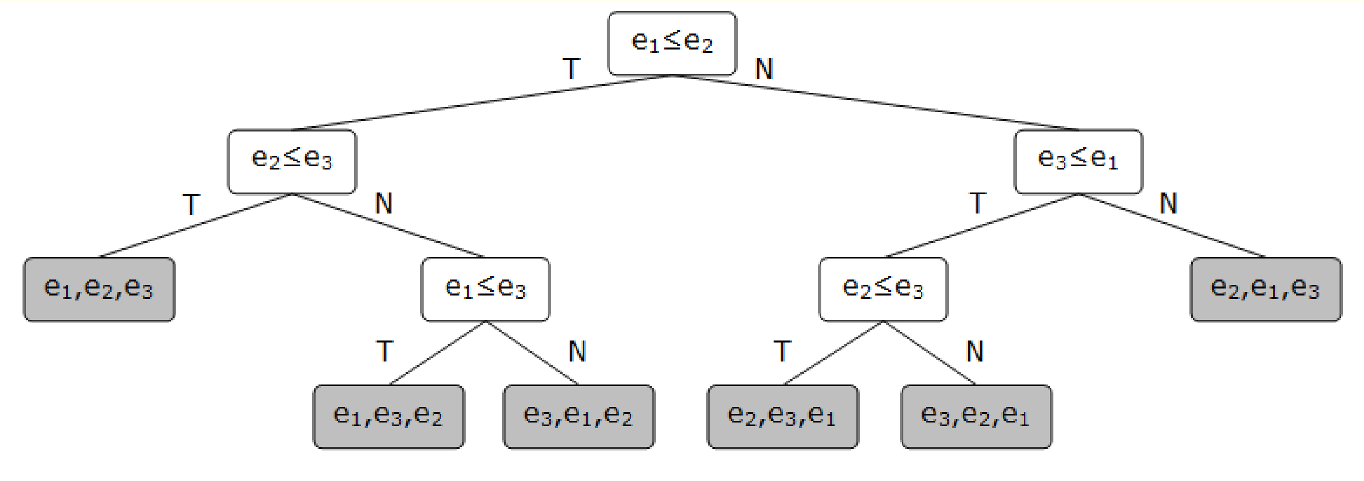
Drzewa decyzyjne i dolne ograniczenie dla problemu sortowania

Przykład. Drzewo decyzyjne algorytmu sortowania przez selekcję dla wejściowego ciągu elementów postaci e_1, e_2, e_3.



Drzewa decyzyjne i dolne ograniczenie dla problemu sortowania

Przykład. Drzewo decyzyjne algorytmu sortowania przez wstawianie dla wejściowego ciągu elementów postaci e_1, e_2, e_3.



Dolne ograniczenie dla problemu sortowania

Drzewa decyzyjne i dolne ograniczenie dla problemu sortowania

Lemat 1. *Niech x_h będzie liczbą liści w drzewie binarnym wysokości h. Wtedy $x_h \leq 2^h$, czyli $h \geq \lfloor \lg x_h \rfloor$.*

Dowód. Indukcja względem h:

- baza indukcji: dla $h = 0$ zachodzi $1 \leq 2^0$, czyli $1 \leq 1$,
- założenie indukcyjne: dla $h = k$ zachodzi $x_k \leq 2^k$,
- teza indukcyjna: dla $h = k + 1$ zachodzi $x_{k+1} \leq 2^{k+1}$,
- dowód tezy: z założenia indukcyjnego $x_k \leq 2^k$, powiększamy drzewo binarne wysokości k do wysokości $k + 1$. W tym celu wybieramy y liści z drzewa wysokości k (na k-tym poziomie drzewa znajduje się co najwyżej 2^k liści) i dodajemy do nich co najwyżej $2y$ wierzchołków następniczych. Zatem $x_{k+1} \leq 2y + 2^k - y = y + 2^k \leq 2^k + 2^k = 2^{k+1}$.

Ostatecznie $x_h \leq 2^h$, czyli $h \geq \lfloor \lg x_h \rfloor$.

Wniosek (z lematu 1). Każde drzewo decyzyjne dla problemu sortowania przez porównania n-elementowego ciągu wejściowego ma wysokość równą co najmniej $\lfloor \lg n! \rfloor = \lfloor n \lg n \rfloor$.

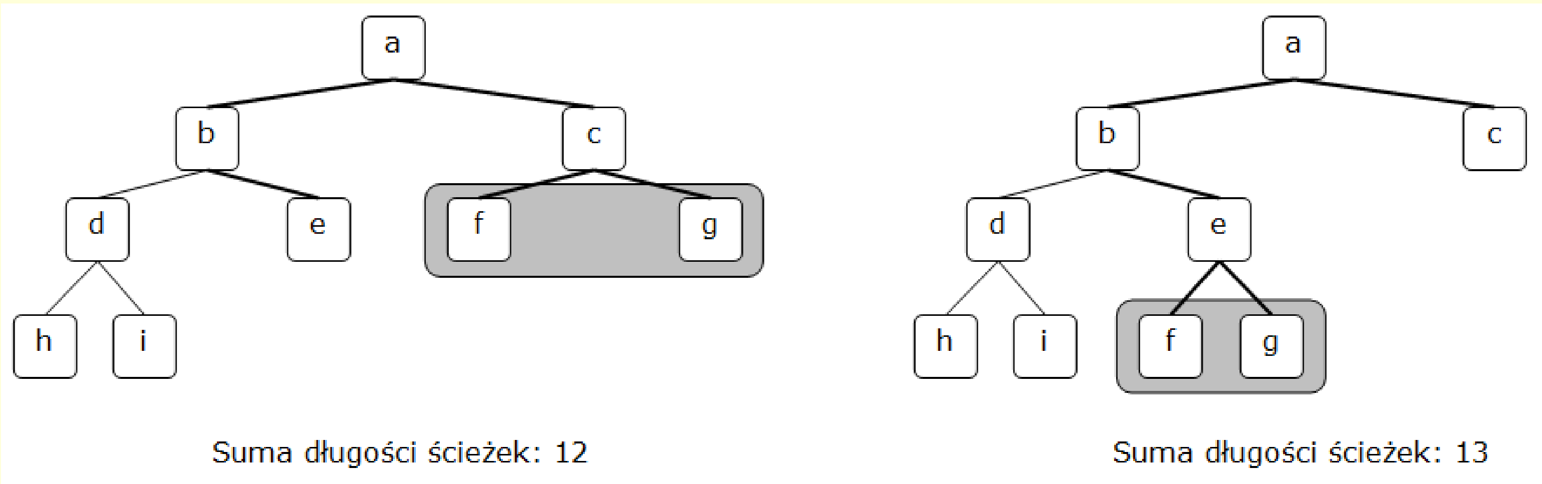
Twierdzenie (z lematu 1). *Dla każdego algorytm sortującego przez porównania n elementowy ciąg wejściowy zachodzi $W(n) = \Omega(n \lg n)$.*

Drzewa decyzyjne i dolne ograniczenie dla problemu sortowania

Lemat 2. *Niech T_x będzie drzewem binarnym lokalnie pełnym o x liściach, $\mathcal{T}_x$ zbiorem wszystkich drzew binarnych lokalnie pełnych o x liściach, a $\sum(T_x)$ sumą długości wszystkich ścieżek korzeń-liść w drzewie T_x, wtedy*

$$\min\left\{\sum(T_x) : T_x \in \mathcal{T}_x\right\} \quad \geq \quad x\lfloor \lg x \rfloor - x.$$

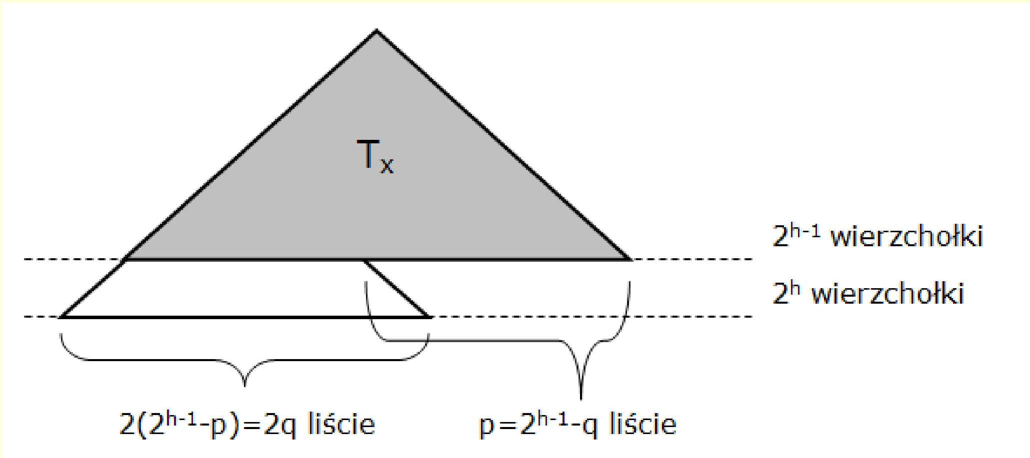
Dowód. Wśród drzew lokalnie pełnych $\mathcal{T}_x$ składających się z x liści, suma długości wszystkich ścieżek korzeń-liść w drzewie T_x jest minimalna wtedy, gdy wszystkie liście drzewa T_x znajdują się na co najwyżej dwóch ostatnich poziomach, np.:



Drzewa decyzyjne i dolne ograniczenie dla problemu sortowania

Dowód. (c.d.) Interesują nas dwa przypadki:

- gdy liście znajdują się jedynie na ostatnim poziomie, wtedy drzewo T_x jest drzewem doskonałym, stąd $\sum(T_x) = x\lfloor \lg x \rfloor \geq x\lfloor \lg x \rfloor - x$,
- gdy liście znajdują się na dwóch ostatnim poziomach, wtedy dla drzewa wysokości h, na poziomie $h-1$ znajdują się $p = 2^{h-1} - q$ liście, a na poziomie h znajdują się $2\left(2^{h-1} - p\right) = 2q$ liście



Drzewa decyzyjne i dolne ograniczenie dla problemu sortowania

Dowód. (c.d.) Zatem

$$\begin{aligned}\sum(T_x) &= p(h-1)+2qh = ph+2qh-p = h(p+2q)-p \\ &= hx-p\end{aligned}$$

i na podstawie lematu 1 (tj. $h \geq \lfloor \lg x_h \rfloor$)

$$\begin{aligned}\sum(T_x) &= hx-p \\ &\geq x\lfloor \lg x \rfloor - p.\end{aligned}$$

Ponieważ dla dowolnego drzewa binarnego lokalnie pełnego zachodzi $p \leq x$, to

$$\sum(T_x) \geq x\lfloor \lg x \rfloor - x.$$

Drzewa decyzyjne i dolne ograniczenie dla problemu sortowania

Lemat 3. *Średnia długość ścieżki w lokalnie pełnym drzewie binarnym o x liściach jest nie mniejsza niż $\lfloor \lg x \rfloor - 1$.*

Dowód. Na podstawie lematu 2, suma ścieżek w przypadku minimalnym w dowolnym lokalnie pełnym drzewie binarnym wynosi co najmniej $x \lfloor \lg x \rfloor - x = x (\lfloor \lg x \rfloor - 1)$, a w **przypadku średnim** $\frac{1}{x} (x (\lfloor \lg x \rfloor - 1)) = \lfloor \lg x \rfloor - 1$.

Wniosek (z lematu 3). Każde drzewo decyzyjne dla problemu sortowania przez porównania n-elementowego ciągu wejściowego ma średnią wysokość równą co najmniej $\lfloor \lg n! \rfloor - 1 = \lfloor n \lg n \rfloor - 1$.

Twierdzenie (z lematu 3). *Dla każdego algorytm sortującego przez porównania n-elementowy ciąg wejściowy zachodzi $A(n) = \Omega(n \lg n)$.*

Wniosek. Algorytm sortowania szybkiego jest optymalnym rozwiązaniem problemu sortowania przez porównania w przypadku oczekiwanym.

Sortowanie przez scalanie

Sortowanie przez scalanie

Założenia. A jest tablicą n różnych liczb naturalnych, gdzie $n = 2^k$ i $k \in \mathbb{N}^+$.

Idea algorytmu sortowania przez scalanie. Niech m będzie rozmiarem aktualnie analizowanego fragmentu tablicy:

- jeżeli $m > 1$, rozdziel w połowie aktualnie rozważany fragment tablicy na dwie podtablice, powtórz rekurencyjnie schemat podziału dla obu podtablic oddzielnie,
- jeżeli $m = 1$, to:
  - przerwij schemat podziału,
  - scalaj rekurencyjnie wszystkie posortowane połowy podtablic, tak aby po każdym kroku scalania aktualnie analizowany fragment tablicy stanowił posortowany ciąg elementów.

Zadanie. Przedstaw działanie algorytmu sortowania szybkiego dla następujących danych wejściowych:

$$A = [7, 5, 3, 2, 12, 17, 8, 4, 11, 15, 16, 19, 20, 1, 0, 13] .$$

Sortowanie przez scalanie

Rozwiązanie. Algorytm sortowania przez scalanie:

```
void Scal(A[n],int l,int r,int m) {
...  // funkcja dokonuje scalenia posortowanych fragmentów tablicy
        A[l], A[l+1], ..., A[m] oraz A[m+1], A[m+2], ..., A[r]
        w jeden posortowany fragment A[l], A[l+1], ..., A[r]
}

void MergeSort(int A[n],int l,int r) { // wp:  n = 2^k i k ∈ N+.
   int m=(l+r) div 2;

   if (r>l) {

      MergeSort(A,l,m);
      MergeSort(A,m+1,r);

      Scal(A,l,m,r);
   }
}
```

Sortowanie przez scalanie – scalanie ciągów uporządkowanych

Założenia. $A[l], A[l+1], \ldots, A[m]$ oraz $A[m+1], A[m+2], \ldots, A[r]$ są niepustymi posortowanymi rosnąco podtablicami tablicy A długości odpowiednio n_1 oraz n_2, gdzie $n_1 + n_2 = n$. Dla uproszczenia przyjmijmy, że $n_1 \leq n_2$.

Idea algorytmu Merge – wariant z pamięcią dodatkową rozmiaru $\Theta(\min(n_1, n_2))$.

- utwórz tablicę pomocniczą Tmp rozmiaru n_1,
- przepisz zawartość podtablicy $A[l], A[l+1], \ldots, A[m]$ do tablicy Tmp,
- niech $i = l$, $w_1 = 0$ oraz $w_2 = m+1$, jeżeli:
  - $w_1 < n_1$ i $w_2 < r$, to jeżeli $Tmp[w_1] < A[w_2]$, to $A[i] = Tmp[w_1]$, zwiększ w_1 oraz i o 1, w p.p. $A[i] = A[w_2]$, zwiększ w_2 oraz i o 1,
  - $w_1 < n_1$ i $w_2 = r$, to $A[i] = Tmp[w_1]$, zwiększ w_1 oraz i o 1,
  - $w_1 = n_1$ i $w_2 < r$, to $A[i] = A[w_2]$, zwiększ w_2 oraz i o 1.

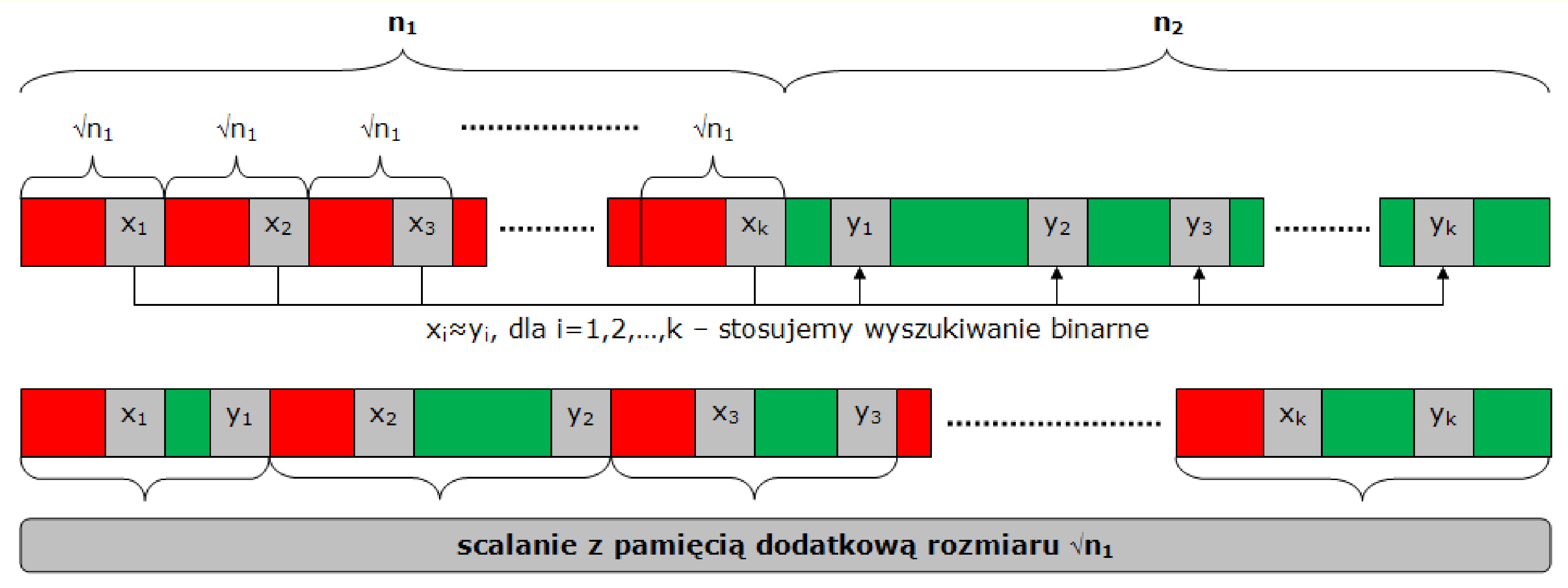
Zadanie. Przedstaw działanie algorytmu Merge dla następujących danych wejściowych:

$$A = [1, 3, 5, 7, 2, 4, 6, 8], l = 0, r = 7, m = 3.$$

Pytanie. Jaka jest złożoność czasowa algorytmu Merge względem liczby porównań?

Sortowanie przez scalanie – scalanie ciągów uporządkowanych

Idea algorytmu Merge – wariant z pamięcią dodatkową rozmiaru $\Theta\left(\sqrt{\min\left(n_1,n_2\right)}\right)$**.**



Złożoność czasowa rozwiązania. Załóżmy, że operacją dominującą jest liczba porównań, wtedy

$$T\left(n\right)=\left\lceil\sqrt{\min\left(n_1,n_2\right)}\right\rceil\cdot O\left(\lg\left(\max\left(n_1,n_2\right)\right)\right)+\Theta\left(n\right)=\Theta\left(n\right).$$

Sortowanie przez scalanie – złożoność algorytmu

Złożoność czasowa algorytmu MergeSort. Niech $T(n)$ będzie liczbą elementarnych operacji porównania elementów sortowanej tablicy jakie wykonuje algorytm sortowania przez scalanie dla danych rozmiaru n, wtedy:

$$T(n) = \begin{cases} 0 & \text{dla } n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{dla } n > 1 \end{cases}.$$

Na podstawie twierdzenia o rekurencji uniwersalnej $T(n) = \Theta(n \lg n)$.

Pytanie. Jaka jest złożoność pamięciowa algorytmu sortowania przez scalanie?

Zadanie (**). Oszacuj złożoność czasową i pamięciową algorytmu scalania opartego na rekurencyjnym wykorzystaniu schematu metody scalania z pamięcią dodatkową rozmiaru $O\left(\sqrt{n}\right)$.

Sortowanie a stabilność i „działanie w miejscu”

Sortowania a stabilność i „działanie w miejscu”

Definicja. Algorytm sortowania nazywamy *stabilnym* wttw, gdy wszystkie powtarzające się elementy w ciągu wyjściowym występują w niezmienionej kolejności w odniesieniu do ciągu wejściowego.

Pytanie. Który z omawianych do tej pory algorytmów sortujących jest stabilny?

Definicja. Algorytm sortowania *„działa w miejscu”* wttw, gdy złożoność pamięciowa algorytmu jest stała.

Pytanie. Który z omawianych do tej pory algorytmów sortujących „działa w miejscu”?

Sortowanie w czasie liniowym

(algorytm sortowania kubełkowego)

Sortowanie w czasie liniowym – algorytm sortowania kubełkowego

Założenie. Niech A będzie tablicą n liczb rzeczywistych z przedziału $[0, 1)$ wygenerowanych z rozkładem jednostajnym.

Idea algorytmu sortowania kubełkowego.

- utwórz n kubełków $b_0, b_1, \ldots, b_{n-1}$,
- rozmieść w kubełkach wszystkie elementy tablicy A tak, że do kubełka b_i trafią elementy $A[j]$ takie, że $i \cdot \frac{1}{n} \leq A[j] < (i+1) \cdot \frac{1}{n}$,
- posortuj każdy kubełek oddzielnie metodą sortowania przez wstawianie,
- połącz kolejno wszystkie kubełki w ciąg wynikowy.

Zadanie. Przedstaw działanie algorytmu sortowania kubełkowego dla dowolnych 10-ciu liczb rzeczywistych z przedziału $[0, 1)$.

Sortowanie w czasie liniowym – algorytm sortowania kubełkowego

Rozwiązanie. Algorytm sortowania kubełkowego:

```
void BucketSort(real A[],int n) { // wp:  ∀0 ≤ i < n : A[i] ∈ [0,1)
   int i;
   Bucket B[n]; // utworzenie i inicjalizacja kubełków

   for (i=0;i<n;i++)
      Wstaw(A[i],B[⌊nA[i]⌋]); // wstawianie liczby do kubełka

   for (i=0;i<n;i++)
      InsertionSort(B[i]); // sortowanie kubełka

   A=Połącz_kubełki(B);
}
```

Pytanie. Jaka jest złożoność czasowa algorytmu sortowania kubełkowego w przypadku średnim?

Pytanie. Jaka jest złożoność czasowa algorytmu sortowania kubełkowego w przypadku pesymistycznym?

Pytanie. Jaka jest złożoność pamięciowa algorytmu sortowania kubełkowego?

Sortowanie w czasie liniowym

(algorytm sortowania pozycyjnego)

Sortowanie w czasie liniowym – algorytm sortowania pozycyjnego

Założenie. Niech A będzie tablicą n obiektów $o_1, o_2, \ldots, o_{n-1}$, z których każdy zbudowany jest z d elementów $e_{o_i,0}, e_{o_i,1}, \ldots, e_{o_i,d-1}$, gdzie $0 < i < n-1$, które to elementy należą do pewnego uniwersum rozmiaru k np. liczby i cyfry, słowa i litery, itd.

Idea algorytmu sortowania pozycyjnego. Utwórz k kubełków, dla każdego elementu uniwersum oddzielny kubełek. Niech $r = d-1$, d-krotnie powtórz następujący schemat działania:

- rozrzuć obiekty $o_1, o_2, \ldots, o_{n-1}$ do kubełków względem r-tego elementu $e_{o_i,r}$, gdzie $0 < i < n-1$,
- połącz kubełki w ciąg wynikowy,
- zmniejsz r o jeden.

Zadanie. Przedstaw działanie algorytmu sortowania pozycyjnego dla dowolnych 10-ciu 3-cyfrowych liczb naturalnych.

Sortowanie w czasie liniowym – algorytm sortowania pozycyjnego

Rozwiązanie. Algorytm sortowania pozycyjnego:

```
void RadixSort() {
   int r;

   for (r=d-1;r>=0;r--) {
      Rozrzuć obiekty do kubełków względem elementu r-tego;
      Połącz kubełki w ciąg wynikowy;
   }
}
```

Pytanie. Jaka jest złożoność czasowa algorytmu sortowania pozycyjnego w przypadku średnim?

Pytanie. Jaka jest złożoność czasowa algorytmu sortowania pozycyjnego w przypadku pesymistycznym?

Pytanie. Jaka jest złożoność pamięciowa algorytmu sortowania pozycyjnego?

Pytanie. Jak rozrzucać obiekty do kubełków i następnie łączyć kubełki, aby algorytm sortowania pozycyjnego miał własność stabilności?

Sortowanie w czasie liniowym

(algorytm sortowania przez zliczanie)

Sortowanie w czasie liniowym – algorytm sortowania przez zliczanie

Założenie. Niech A będzie tablicą n liczb naturalnych z przedziału $[0, k)$, gdzie $k \ll n$.

Idea algorytmu sortowania przez zliczanie.

- dla każdej liczby $x \in A$ wyznacz liczbę c_x liczb $y \in A$ takich, że $y \leq x$,
- umieść każdą z liczb $x \in A$ w tablicy wyjściowej na pozycji c_x.

Zadanie. Przedstaw ideę działanie algorytmu sortowania przez zliczanie dla danych wejściowych:

$$A = [4, 3, 2, 1, 0, 5, 0, 2, 2, 1, 3, 2, 4, 1] .$$

Sortowanie w czasie liniowym – algorytm sortowania przez zliczanie

Rozwiązanie. Algorytm sortowania przez zliczanie:

```
void CountingSort(int A[], int n) {
   int i;
   int Tmp[k], Result[n];

   for (i=0;i<n;i++) // zliczenie liczb równych
      Tmp[A[i]]=Tmp[A[i]]+1;

   for (i=1;i<k;i++) // zliczenie liczb mniejszych równych
      Tmp[i]=Tmp[i]+Tmp[i-1];

   for (i=n-1;i>=0;i--) { // ostateczne rozmieszczenie liczb
      Result[Tmp[A[i]]-1]=A[i];
      Tmp[A[i]]=Tmp[A[i]]-1;
   }
}
```

Zadanie. Przedstaw działanie algorytmu sortowania przez zliczanie zgodnie z zaprezentowaną implementacją, dla danych wejściowych:

$$A = [4, 3, 2, 1, 0, 5, 0, 2, 2, 1, 3, 2, 4, 1] .$$

Sortowanie w czasie liniowym – algorytm sortowania przez zliczanie

Pytanie. Czy algorytm CountingSort zgodny z zaprezentowaną implementacją ma własność stabilności?

Pytanie. Jaka jest złożoność czasowa algorytmu sortowania przez zliczanie w przypadku średnim?

Pytanie. Jaka jest złożoność czasowa algorytmu sortowania przez zliczanie w przypadku pesymistycznym?

Pytanie. Jaka jest złożoność pamięciowa algorytmu sortowania przez zliczanie?