

Wstęp do informatyki

Wykład 8

Dziel i zwyciężaj

Instytut Informatyki UWr

Temat wykładu

Metoda dziel i zwyciężaj:

- max & min (jednocześnie)
- sortowanie przez scalanie (merge sort)
- sortowanie szybkie (quick sort)

Dziel i zwyciężaj

1. **Podziel** problem na podproblemy
2. Rozwiąż podproblemy
3. **Połącz** rozwiązania podproblemów tak, aby uzyskać rozwiązanie problemu głównego

Przykład.

Wyszukiwanie binarne!

Max & min problem

Specyfikacja

Wejście: $n, a[0..n-1]$

Wyjście: min – najmniejszy element $a[0..n-1]$
max – największy element $a[0..n-1]$

Rozw. 1

- Oblicz min
- Oblicz max
- Zwróć (min, max)

Max & min problem – v.1

Rozw. 1

- Oblicz min
- Oblicz max
- Zwróć(min, max)

```
...  
int min, max; //min, max globalne  
...  
void maxmin(int a[], int n)  
{ int k;  
  max=min=a[0];  
  for (k=1; k<n; k++) {  
    if (min>a[k]) min=a[k];  
    if (max<a[k]) max=a[k];  
  }  
}
```

```
def maxmin1(a, n):  
    max=min=a[0]  
    for k in range(1,n):  
        if min>a[k]: min=a[k]  
        if max<a[k]: max=a[k]  
    return max, min
```

Liczba porównań (najgorszy przypadek): $2n - 2$

Max & min problem – v.1.1

Obserwacja

Jeśli $a[i]$ mniejsze niż najmniejszy element $a[0..i-1]$, nie trzeba sprawdzać czy $a[i]$ większe od największego elementu $a[0..i-1]$.

```
...  
int min, max; //min, max globalne  
...  
void maxmin(int a[], int n)  
{ int k;  
  max=min=a[0];  
  for (k=1; k<n; k++)  
    if (min>a[k]) min=a[k];  
    else if (max<a[k]) max=a[k];  
}
```

```
def maxmin1(a, n):  
    max=min=a[0]  
    for k in range(1,n):  
        if min>a[k]: min=a[k]  
        else:  
            if max<a[k]: max=a[k]  
    return max, min
```

Czas (najgorszy przypadek): $2n - 2$ ☹

Max & min problem – v.2

Algorytm rekurencyjny

- If $n = 1$: $max = min = a[0]$
- If $n = 2$: wybierz min, max z $a[0]$ i $a[1]$ za pomocą jednego porównania
- If $n > 2$:
 - Podziel $a[0..n-1]$ na dwie podtablice równej długości
 - Znajdź $max1, min1$: max & min pierwszej podtablicy
 - Znajdź $max2, min2$: max & min drugiej podtablicy
 - max = największy element w $\{max1, max2\}$
 - min = najmniejszy element w $\{min1, min2\}$

Max & min problem – v.2

Specyfikacja (ogólniej, na potrzeby rozw. rekurencyjnego...)

Wejście:

- l, p – liczby naturalne,
- a – tablica

Wyjście:

- **max** – max w $a[l], a[l+1], \dots, a[p]$
- **min** – min w $a[l], a[l+1], \dots, a[p]$

Max & min problem – v.2

```
int maxmin(int a[], int l, int p)
{ int mi1, mi2, ma1, ma2;
  if (l==p) min = max = a[l];
  else
    if (p-l==1)
      if (a[l]<a[p]) {
        max = a[p]; min = a[l];}
      else
        { max = a[l]; min = a[p];}
    else
    { maxmin(a, l, (l+p)/2);
      mi1 = min; ma1 = max;
      maxmin(a, (l+p)/2+1, p);
      mi2 = min; ma2 = max;
      min = mi1<mi2 ? mi1 : mi2;
      max = ma1>ma2 ? ma1 : ma2;
    }
}
```

Aby znaleźć max i min w
 $a[0], \dots, a[n-1]$:

`maxmin(a, 0, n-1)`

Uwaga:

min i **max** globalne!

Dlaczego?

Max & min problem – v.2

```
def maxmin(a, l, p):  
    if l==p: return a[l],a[l]  
    else:  
        if p-l==1:  
            if a[l]<a[p]: return a[p],a[l]  
            else: return a[l],a[p]  
        else:  
            m1 = maxmin(a, l, (l+p)/2)  
            m2 = maxmin(a, (l+p)/2+1, p)  
            if m1[1]<m2[1] min = m1[1] else min = m2[1]  
            if m1[0]>m2[0] max = m1[0] else max = m2[0]  
            return max, min
```

Aby znaleźć max i min w
 $a[0], \dots, a[n-1]$:

`maxmin(a, 0, n-1)`

Max & min problem – v.2

Liczba porównań :

| n | liczba porównań |
|-----|-----------------|
| 1 | 0 |
| 2 | 1 |
| 4 | 4 |
| 8 | 10 |
| 16 | 22 |
| 32 | 46 |
| 64 | 94 |

#porównań (gdy n parzyste):

$$T(1) = 0$$

$$T(2) = 1$$

$$T(n) = 2T(n/2) + 2 \text{ dla } n > 2, \text{ parz.}$$

Rozwiązanie:

$$T(n) = 3n / 2 - 2 \quad (\text{dla } n=2^k, k>0)$$

mniej niż $2n - 2$

Max & min problem – v.2

#porównań (gdy n parzyste):

$$T(1) = 0$$

$$T(2) = 1$$

$$T(n) = 2T(n/2) + 2$$

Rozwiązanie – **podpowieź**:

$$T(n) = 3/2 n - 2 \text{ dla } n \text{ postaci } n=2^k$$

Pyt.: jak sprawdzić poprawność **podpowiezi**?

Odp.: indukcja matematyczna.

Max & min problem – v.2

#porównań (gdy n parzyste):

$$T(1) = 0$$

$$T(2) = 1$$

$$T(n) = 2T(n/2) + 2$$

Jak poradzić sobie bez **podpowiedzi**?

Rozwiązanie metodą podstawienia (dla $n=2^k$, $k>0$) :

$$\begin{aligned} T(n) &= 2 T(n/2) + 2 = 2 (2 T(n/4) + 2) + 2 = \\ &= 4 T(n/4) + 4 + 2 = 4 (2 T(n/8) + 2) + 4 + 2 = \\ &= 8 T(n/8) + 8 + 4 + 2 = \dots = \\ &= 2^i T(n/2^i) + 2^i + 2^{i-1} + \dots + 2^2 + 2^1 = \dots = \\ &= 2^{k-1} T(n/2^{k-1}) + 2^{k-1} + 2^{k-2} + \dots + 2^2 + 2^1 = \\ &= n/2 T(2) + n/2 + n/4 + \dots + 4 + 2 \\ &= n/2 + n - 2 = 3/2 n - 2 \end{aligned}$$

// $i \leq k$

Uwaga: korzystamy ze znanych wzorów na sumę ciągu geometrycznego, ...
czyli z tożsamości $1 + 2 + 4 + \dots + 2^{k-1} = 2^k - 1$

Sortowanie przez scalanie (merge sort)

Scalanie - przypomnienie

Specyfikacja:

Wejście:

- n, m – liczby naturalne
- $a[0] \leq a[1] \leq \dots \leq a[n - 1]$
- $b[0] \leq b[1] \leq \dots \leq b[m - 1]$

Wyjście:

- $c[0] \leq c[1] \leq \dots \leq c[n+m - 1]$
- elementy ciągu c to „suma” (z powtórzeniami) elementów ciągów a i b

Scalanie – przykład

Przykład:

We:

7 5

3 5 5 7 8 8 9

3 4 6 12 14

n m

$a[0], \dots, a[n-1]$

$b[0], \dots, b[m-1]$

Wy:

3 3 4 5 5 6 7 8 8 9 12 14

Scalanie - przypomnienie

Algorytm z poprzedniego wykładu:

Czas : $O(n+m)$

Pamięć : $O(n+m)$

Scalanie – inna specyfikacja

Wejście:

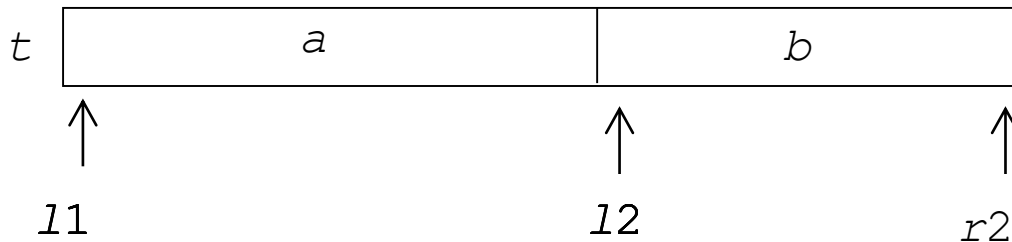
- Tablica `int t[]`
- Liczby $l1$, $l2$, $r2$

takie że

- $t[l1..l2-1]$ posortowane (odpowiada a)
- $t[l2..r2]$ posortowane (odpowiada b)

Wynik:

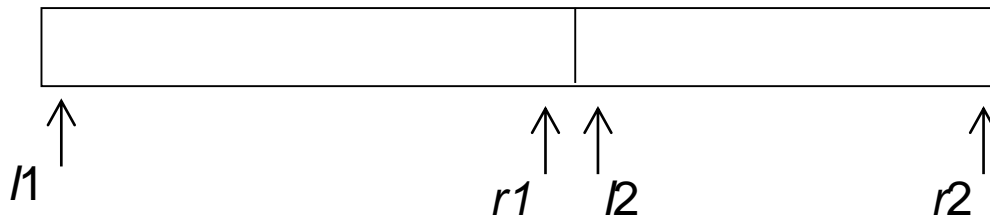
$t[l1..r2]$ posortowane (zawiera sumę a i b)



Scalanie – inna specyfikacja

Algorytm:

1. $i \leftarrow l1, j \leftarrow l2, k \leftarrow 0, r1 \leftarrow l2 - 1$ // i, j – „strażnicy”
2. dopóki ($i \leq r1$ oraz $j \leq r2$)
 - jeżeli ($t[i] < t[j]$) // wybór najmn. jeszcze poza c
 - $c[k] \leftarrow t[i], i \leftarrow i+1$
 - w przeciwnym przypadku:
 - $c[k] \leftarrow t[j], j \leftarrow j+1$
 - $k \leftarrow k+1$
3. dopóki ($i \leq r1$) // kopiowanie „ogona” pierwszego ciągu
 - $c[k] \leftarrow a[i], i \leftarrow i+1, k \leftarrow k+1$
4. dopóki ($j \leq r2$) // kopiowanie „ogona” drugiego ciągu
 - $c[k] \leftarrow b[j], j \leftarrow j+1, k \leftarrow k+1$
5. Skopiuj $c[0..k-1]$ do $t[l1..r2]$



Scalanie – inna specyfikacja

Implementacja

```
merge(int t[], int l1, int l2, int r2)
{ int k,i,j,c[n],r1;

    k=0;
    r1=l2; r2++;
    while (l1<r1 && l2<r2)
        c[k++]=(t[l1]<t[l2])? t[l1++]:t[l2++];
    while (l1<r1) c[k++]=t[l1++];
    while (l2<r2) c[k++]=t[l2++];
    while (k>0) t[--r2]=c[--k];
}
```

Scalanie – inna specyfikacja

Implementacja

```
def merge(t, l1, l2, r2):  
    c=Array(r2-l1+1)  
    k=0; r1=l2; r2+=1  
    while l1<r1 and l2<r2:  
        if t[l1]<t[l2]:  
            c[k]=t[l1]; l1+=1  
        else:  
            c[k]=t[l2]; l2+=1  
        k+=1  
    while l1<r1:  
        c[k]=t[l1]; k+=1; l1+=1  
    while l2<r2:  
        c[k]=t[l2]; k+=1; l2+=1  
    while k>0:  
        r2-=1; k-=1; t[r2]=c[k]
```

Alg. scalania – złożoność raz jeszcze

Rozmiar danych: $n + m$

Czas : $O(n+m)$

Pamięć : $O(n+m)$

UWAGA: potrzebujemy „pomocniczej”
tablicy c rozmiaru $n + m$.

Sortowanie przez scalanie

Specyfikacja

Dane:

- n – liczba naturalna dodatnia
- $a[0..n - 1]$ – tablica elementów ze zbioru uporządkowanego

Wynik: elementy $a[0.. n - 1]$ w porządku niemalejącym, czyli

$$a[0] \leq a[1] \leq \dots \leq a[n - 1].$$

Sortowanie przez scalanie (zasada „dziel i zwyciężaj”)

1. Podziel ciąg wejściowy A na dwa podciągi A_1 , A_2 (prawie) tej samej długości
2. Posortuj A_1 i A_2 osobno (rekurencja)
3. Scal posortowane ciągi A_1 i A_2

Sortowanie przez scalanie dokładniej

Algorytm (idea):

- Jeśli $n=1$: zwróć tablicę a
- Podziel $a[0..n-1]$ na podtablice $a[0..k]$, $a[k+1..n-1]$ o (prawie) równej długości
- Posortuj $a[0..k]$ //rekurencja!
- Posortuj $a[k+1..n-1]$ //rekurencja!
- Scal $a[0..k]$ oraz $a[k+1..n-1]$
- Zwróć a jako wynik

Sortowanie przez scalanie

Specyfikacja ogólniej

Dane:

- l, r – liczby naturalne takie, że $l \leq r$
- a – tablica nad zbiorem uporządkowanym Z

Wynik: elementy $a[l], \dots, a[r]$ w porządku niemalejącym, czyli

$$a[l] \leq a[l+1] \leq \dots \leq a[r].$$

Sortowanie przez scalanie dokładniej

Algorytm ogólniej (idea):

- Jeśli $l=r$: zakończ
- Podziel $a[l..r]$ na podtablice $a[l..s]$, $a[s+1..r]$ o (prawie) równej długości
- Posortuj $a[l..s]$ //rekurencja!
- Posortuj $a[s+1..r]$ //rekurencja!
- Scal $a[l..s]$ oraz $a[s+1..r]$
- Zwróć $a[l..r]$ jako wynik

Sortowanie przez scalanie

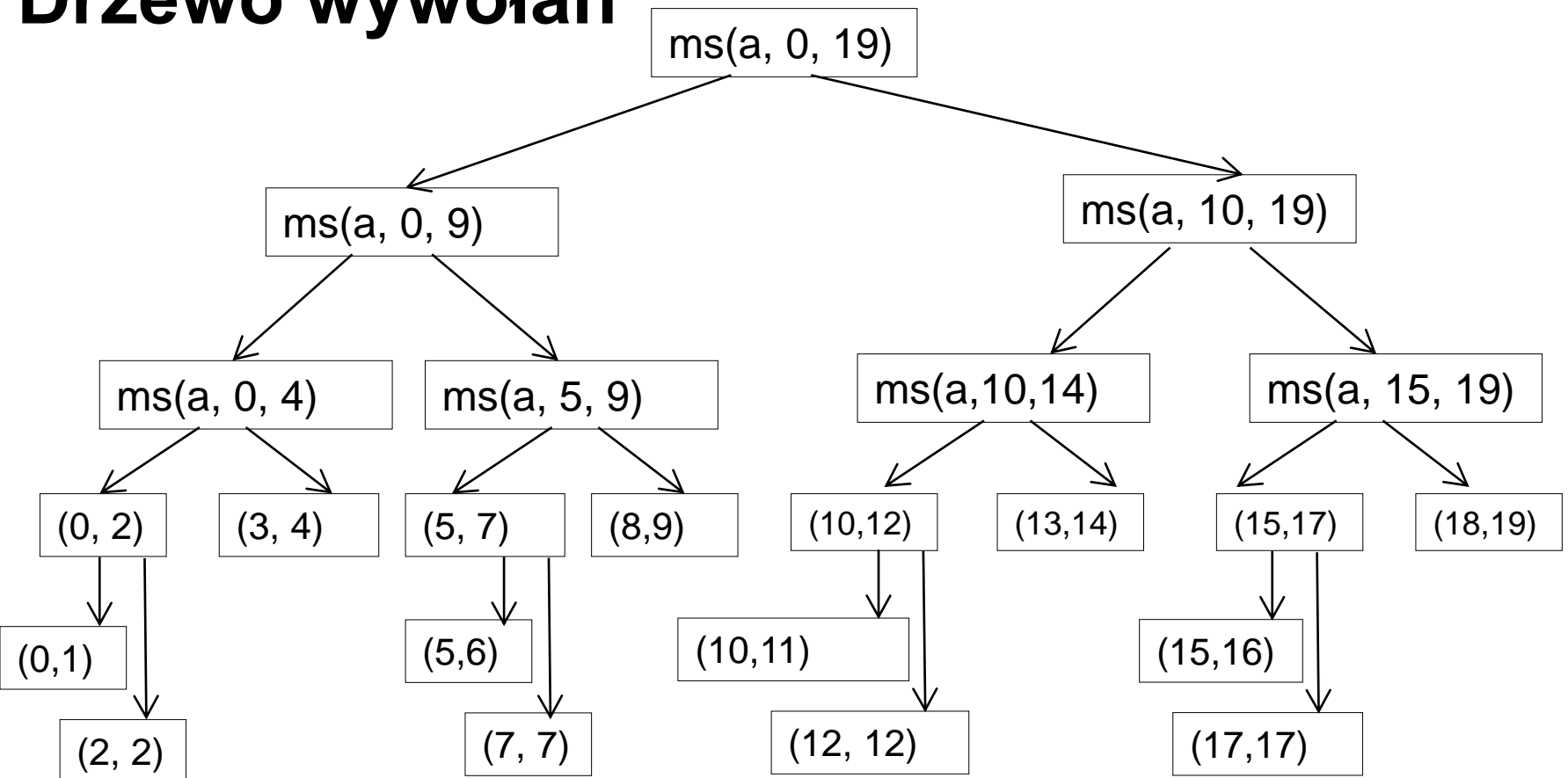
Implementacja:

```
mergesort(int a[], int l, int r)
//sortuje a[l..r]
{ int s;
  if (r-l>0)
  { s=(l+r)/2;
    mergesort(a,l,s);
    mergesort(a,s+1,r);
    merge(a,l,s+1,r);
  }
}
```

```
def mergesort(a, l, r):
#sortuje a[l..r]
    if r-l>0:
        s=(l+r)/2
        mergesort(a,l,s)
        mergesort(a,s+1,r)
    merge(a,l,s+1,r)
```

Sortowanie przez scalanie

Drzewo wywołań



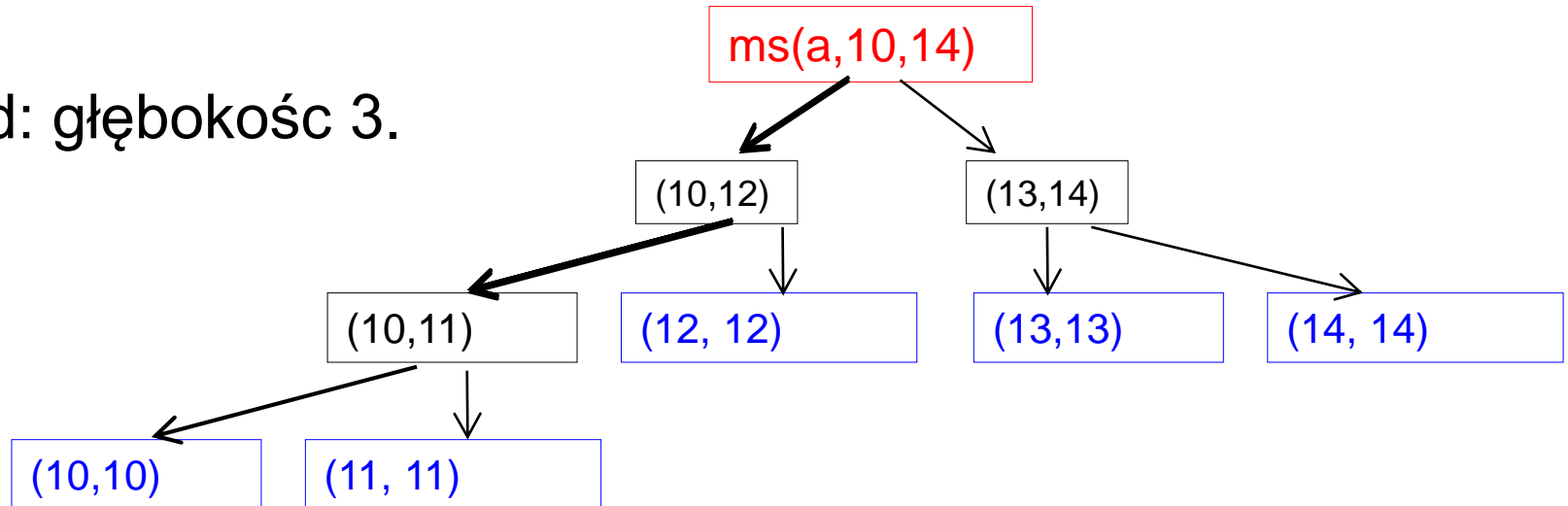
Uwaga: wywołania dla niektórych ciągów o długości ≤ 2 nie zmieściły się.

Sortowanie przez scalanie

Drzewo wywołań:

- **korzeń** – wywołanie pierwotne (np. $ms(a, 0, 19)$)
- **liść** – wywołanie, z którego nie ma wywołań rekurencyjnych (np. $ms(a, 10, 10)$)
- głębokość drzewa - długość najdłuższej **ścieżki** od korzenia do liścia

Przykład: głębokość 3.



Sortowanie przez scalanie - poprawność

Indukcja względem różnicy $r - l$:

- Jeśli $r - l = 0$: jedno-elementowy ciąg do posortowania, funkcja kończy działanie;
- **Zał.:** poprawność dla $r - l < n$

Teza: poprawność dla $r - l = n$, $n > 0$

Dowód:

- wówczas s spełnia $l \leq s < r$, czyli ...
- ... rekurencyjne wywołania na krótszych podciągach $a[l..s]$ i $a[s+1..r]$ działają poprawnie na mocy założenia indukcyjnego
- a scalanie poprawnie połączy posortowane podciągi (p. poprzedni wykład)

Sortowanie przez scalanie - złożoność

Pamięć ($n = r - l + 1$)

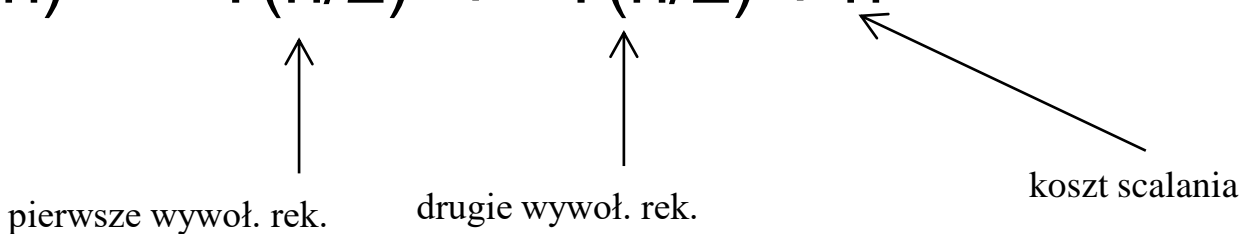
- $n + O(n)$ + **stos wywołań**
- Jak (oszczędnie) zaalokować tablicę pomocniczą?

Sortowanie przez scalanie - złożoność

Pamięć ($n = r - l + 1$)

- $n + O(n)$ + **stos wywołań**
- Jak (oszczędnie) zaalokować tablicę pomocniczą?

Czas: ($n = r - l + 1$)

- $T(n) = T(n/2) + T(n/2) + n$ 

pierwsze wywoł. rek. drugie wywoł. rek. koszt scalania

Rozwiązanie: $T(n) \cong n \log n$

Zależność rekurencyjna

Czas merge sort dokładniej:

- $T(1) = 1$
- $T(n) = T(n/2) + T(n/2) + n$ dla parzystego $n > 1$
- $T(n) = T((n+1)/2) + T((n-1)/2) + n$ dla nieparzystego $n > 1$

Zależność rekurencyjna

Czas merge sort:

- $T(1) = 1$
- $T(n) = T(n/2) + T(n/2) + n$ dla parzystego $n > 1$

Rozwiązanie metoda podstawienia

(zakładamy, że $n=2^k$ dla naturalnego k , czyli $k = \log n$):

$$\begin{aligned} T(n) &= 2 T(n/2) + n = 2 (2 T(n/4) + n/2) + n = \\ &= 4 T(n/4) + 2n = 4 (2 T(n/8) + n/4) + 2n = \\ &= 8 T(n/8) + 3 n = \dots = \\ &= 2^i T(n/2^i) + i n = \dots = \quad // i \leq k \\ &= 2^k T(n/2^k) + k n = n T(1) + n \log n \\ &= n + n \log n = O(n \log n) \end{aligned}$$

Sortowanie szybkie (quick sort)

Quicksort – kolejny przykład zastosowania metody dziel i zwyciężaj

1. Wybierz element x ciągu wejściowego A i podziel A na dwa rozłączne podciągi:
 - $A1$: elementy nie większe niż x
 - $A2$: elementy nie mniejsze niż x
2. Posortuj $A1$ i $A2$ osobno (rekurencja)
3. Zwróć złączone ciągi $A1$ i $A2$

Uwaga

*Różne wystąpienia elementu x mogą należeć do $A1$ lub $A2$, **ALE** każda kopia należy tylko do jednego ze zbiorów $A1$, $A2$.*

Specyfikacja – przypomnienie

Specyfikacja (na potrzeby „dziel i zwyciężaj”)

Dane:

- l, r – liczby naturalne takie, że $l \leq r$
- a – tablica nad zbiorem uporządkowanym Z

Wynik: elementy $a[l], \dots, a[r]$ w porządku niemalejącym, czyli

$$a[l] \leq a[l+1] \leq \dots \leq a[r].$$

Quick sort dokładniej...

Jeśli $l < r$:

1. Wybierz x (**pivot**) ze zbioru $\{a[l], a[l+1], \dots, a[r]\}$ i przemieść elementy $a[l], a[l+1], \dots, a[r]$ tak, aby:

- Elementy $a[l..s]$ były $\leq x$
- Elementy $a[s+1..r]$ były $\geq x$

gdzie s jest takie, że $l \leq s < r$

... czyli wykonaj algorytm podziału z poprzedniego wykładu

2. Posortuj: **// rekurencja**

- $a[l..s]$
- $a[s+1..r]$

Podział - przypomnienie

Specyfikacja:

We :

- l, r – liczby naturalne takie, że $l < r$
- a – tablica elementów nad zbiorem uporz.
- $x = a[l]$ (pivot)

Wy :

- s – liczba naturalna, $l \leq s < r$
- (multi)zbiór $\{a[l], \dots, a[r]\}$ bez zmian, ale w takiej kolejności, że
 - $a[l] \leq x, a[l+1] \leq x, \dots, a[s] \leq x$
 - $a[s+1] \geq x, a[s+2] \geq x, \dots, a[r] \geq x$

Podział – implementacja - przypomnienie

```
podzial(int l, int r, int a[])
{ int i,j,y;
  x=a[l]; i=l; j=r;
  while (i<j)
  { while (a[j]>x) j--;
    while (a[i]<x) i++;
    if (i<j)
    { y=a[j]; a[j]=a[i];
      a[i]=y;
      i++; j--; }
  }
  return j;
}
```

```
def podzial(l,r,a):
    x=a[l]
    i=l
    j=r
    while i<j:
        while a[j]>x: j-=1
        while a[i]<x: i+=1
        if i<j:
            y=a[j]
            a[j]=a[i]
            a[i]=y
            i+=1
            j-=1
    return j
```

Quicksort - implementacja

```
qsort(int a[], int l, int r)
//sortuje a[l...r]
{ int s;
  if (l < r) {
    s = podzial(l, r, a);
    qsort(a, l, s);
    qsort(a, s+1, r);
  }
}
```

```
def qsort(a, l, r):
#sortuje a[l...r]
    if (l < r):
        s = podzial(l, r, a)
        qsort(a, l, s)
        qsort(a, s+1, r);
```

Quicksort - poprawność

Indukcja względem różnicy $r - l$:

- Jeśli $r - l \leq 0$: ciąg do posortowania pusty lub jednoelementowy, funkcja kończy działanie;
- **Zał.:** poprawność dla $r - l < n$

Teza: poprawność dla $r - l = n$, $n > 0$

Dowód:

- **podzial** działa poprawnie (p. poprzedni wykład) – podzieli $a[l..r]$ na $X=a[l..s]$ i $Y=a[s+1..r]$ takie, że każdy element Y jest większy (bądź równy) każdemu elementowi X
- gdy s spełnia $l \leq s < r$, to:
rekurencyjne wywołania na krótszych podciągach $a[l..s]$ i $a[s+1..r]$ działają poprawnie na mocy **założenia indukcyjnego**

Quicksort - poprawność

Dowód c.d.:

- gdy $s = r$, to ...
- ... mamy problem: `qsort(a, l, r)` wywoła `qsort(a, l, r)`, które wywoła `qsort(a, l, r)` itd. pętla nieskończona
- ale funkcja `podzial` zwraca s takie, że: $l \leq s < r$
(jako wartość zwracamy j , którego początkowa wartość jest równa r , ponadto wartość j jest w funkcji `podzial` co najmniej raz zmniejszana - p. poprzedni wykład)

Quicksort: złożoność pamięciowa

Pamięć:

$n + O(1) + \text{stos wywołań}$

- Tablica zawierająca ciąg wejściowy
- Stała liczba dodatkowych zmiennych
- Pamięć potrzebna do realizacji rekurencji...?

Quicksort – podział – rola pivota

Pyt.: a gdybyśmy chcieli, żeby pivot nie był pierwszym elementem ciągu?

Odp.: zamiana pivota z pierwszym elementem przed wywołaniem funkcji **podzial**

Pyt.: czy warto wybrać inny pivot? losowy?

Wskazówka: jaki podział jest najkorzystniejszy w metodzie dziel i zwyciężaj? (jakie długości podciągów?)

Quicksort: złożoność czasowa

Najgorszy przypadek: funkcja **podział** dzieli problem na podproblemy o rozmiarach 1 i $n - 1$:

$$T(1) = 1$$

$$T(n) = T(n - 1) + T(1) + n \quad \text{dla } n > 1$$

Pierwsze wywoł rekur.

Drugie wywoł rekur.

Koszt podziału

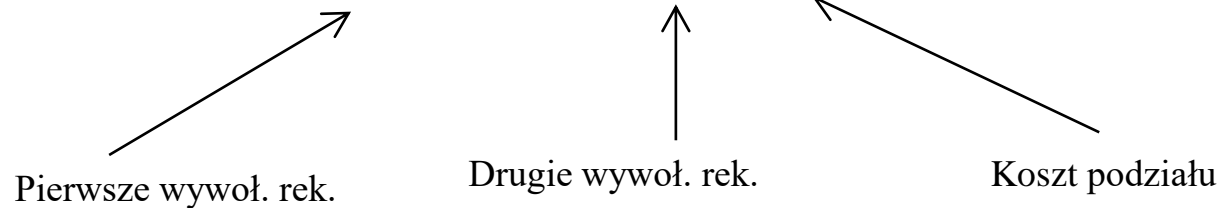
$$\text{Rozwiązanie: } T(n) \cong n^2/2 = O(n^2)$$

Quicksort: złożoność czasowa

- „Zamierzona” sytuacja: procedura **podział** dzieli problem rozmiaru n na dwa podproblemy rozmiaru $n/2$:

$$T(1) = 1$$

$$T(n) = T(n/2) + T(n/2) + n \quad \text{dla } n > 1$$



Rozwiązanie: $T(n) = O(n \log n)$

Quicksort: złożoność czasowa

PRAKTYKA: Quicksort jest najszybszym algorytmem sortowania.

Dlaczego?

- **Średni** czas działania Quicksort jest $O(n \log n)$
„Średnia” z czasów działania algorytmu dla wszystkich permutacji ciągu różnych elementów
- Możliwe jest uzyskanie **gwarantowanego** czasu $O(n \log n)$ w najgorszym przypadku
Korzystamy wówczas z algorytmu szukającego mediany („środkowego” elementu) w czasie $O(n)$.
P. algorytm magicznych piątek.

Quicksort: złożoność czasowa i pamięciowa

PRAKTYKA: Quicksort jest najszybszym algorytmem sortowania.

Porównanie z mergesort:

- Stałe „mnożnikowe” w złożoności czasowej („średniej”) quicksort są mniejsze niż np. w mergesort
- Mniejsza pamięć niż w mergesort (poniżej pomijamy stos wywołań funkcji):

Quicksort: $n + O(1)$

Mergesort $n + n$