Metoda Dziel i Zwyciężaj.

SCHEMAT OGÓLNY

function $DiZ(x)$
  if x jest małe lub proste **then** **return** $AdHoc(x)$
  przekształć x na $x_1, \ldots, x_k$ o mniejszym rozmiarze niż x
  for $i \leftarrow 1$ **to** k **do** $y_i \leftarrow DiZ(x_i)$
  na podstawie $y_1 \ldots y_k$ oblicz rozwiązanie y dla x
  return y

MASTER THEOREM

Twierdzenie 1 *Niech $a, b, c \in N$. Rozwiązaniem równania rekurencyjnego*

$$T(n) = \begin{cases} b & \text{dla } n = 1 \\ aT(n/c) + bn & \text{dla } n > 1 \end{cases}$$

dla n będących potęgą liczby c jest

$$T(n) = \begin{cases} \Theta(n) & \text{jeżeli } a < c, \\ \Theta(n \log n) & \text{jeżeli } a = c, \\ \Theta(n^{\log_c a}) & \text{jeżeli } a > c \end{cases}$$

procedure mergesort($T[1..n]$)
  if n jest małe **then** insert(T)
  else
    $X[1 \,..\, \lceil n/2 \rceil] \leftarrow T[1 \,..\, \lceil n/2 \rceil]$
    $Y[1 \,..\, \lfloor n/2 \rfloor] \leftarrow T[1 + \lceil n/2 \rceil \,..\, n]$
    mergesort(X); mergesort(Y)
    $T \leftarrow$ merge(X, Y)

$t(n) = t(\lceil n/2 \rceil) + t(\lfloor n/2 \rfloor) + \Theta(n) = \Theta(n \log n)$

$MinMax1(S)$
  $S_m \leftarrow S_M \leftarrow \emptyset$
  for $i = 1$ **to** n **div** 2 **do**
    porównaj a_i z a_{n-i+1}; mniejszą z tych liczb wstaw do zbioru S_m, a większą - do zbioru S_M
  $m \leftarrow min\{a \mid a \in S_m\}$
  $M \leftarrow max\{a \mid a \in S_M\}$
  if n parzyste **then** **return** (m, M)
    else **return** $(min(m, a_{\lceil n/2 \rceil}), max(M, a_{\lceil n/2 \rceil}))$

Fakt 3 *Algorytm MinMax1 wykonuje $\lceil \frac{3}{2}n - 2 \rceil$ porównań na elementach zbioru S.*

ALGORYTM KARACUBY = $O(n^{\log 3})$

$multiply(a, b)$
  $n \leftarrow max(|a|, |b|)$ (* $|x|$ oznacza długość liczby x *)
  if n jest małe **then** pomnóż a i b klasycznym algorytmem
    return obliczony iloczyn
  $p \leftarrow \lfloor n/2 \rfloor$
  $a_1 \leftarrow \lfloor a/2^p \rfloor$; $a_0 \leftarrow a$ **mod** 2^p
  $b_1 \leftarrow \lfloor b/2^p \rfloor$; $b_0 \leftarrow b$ **mod** 2^p
  $z \leftarrow multiply(a_0, b_0)$
  $y \leftarrow multiply(a_1 + a_0, b_1 + b_0)$
  $x \leftarrow multiply(a_1, b_1)$;
  return $2^{2p}x + 2^p(y - x - z) + z$

$$T(n) = \begin{cases} k & \text{dla } n = 1 \\ 3T(n/2) + \Theta(n) & \text{dla } n > 1 \end{cases}$$

procedure Quicksort($T[1..n]$)
  if n jest małe **then** insert(T)
  else
    wybierz element dzielący x
    (* niech k równa się liczbie elementów tablicy T nie większych od x*)
    przestaw elementy tablicy T tak, że $\forall_{i \leq k} T[i] \leq x$
    Quicksort($T[1..k]$); Quicksort($T[(k+1)..n]$);

3.5 Para najbliżej położonych punktów

PROBLEM:

Dane: Zbiór $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ współrzędnych punktów na płaszczyźnie.

Zadanie: Znaleźć dwa najbliżej położone względem siebie punkty w P, tj. znaleźć i,j takie, że $d(x_i, y_i, x_j, y_j) = min\{d(x_k, y_k, x_l, y_l) \mid 1 \le k < l \le n\}$, gdzie $d(x_k, y_k, x_l, y_l) = \sqrt{(x_k - x_l)^2 + (y_k - y_l)^2}$.

3.5.1 Strategia Dziel i Zwyciężaj

1. (a) Sortujemy punkty z P według współrzędnych x i zapamiętujemy je w tablicy X;

   (b) Sortujemy punkty z P według współrzędnych y i zapamiętujemy je w tablicy Y;

   (c) Znajdujemy prostą l dzielącą P na dwa równoliczne (z dokładnością do 1) podzbiory:

   - P_L- podzbiór punktów leżących na lewo od l,
   - P_R - podzbiór punktów leżących na prawo od l.

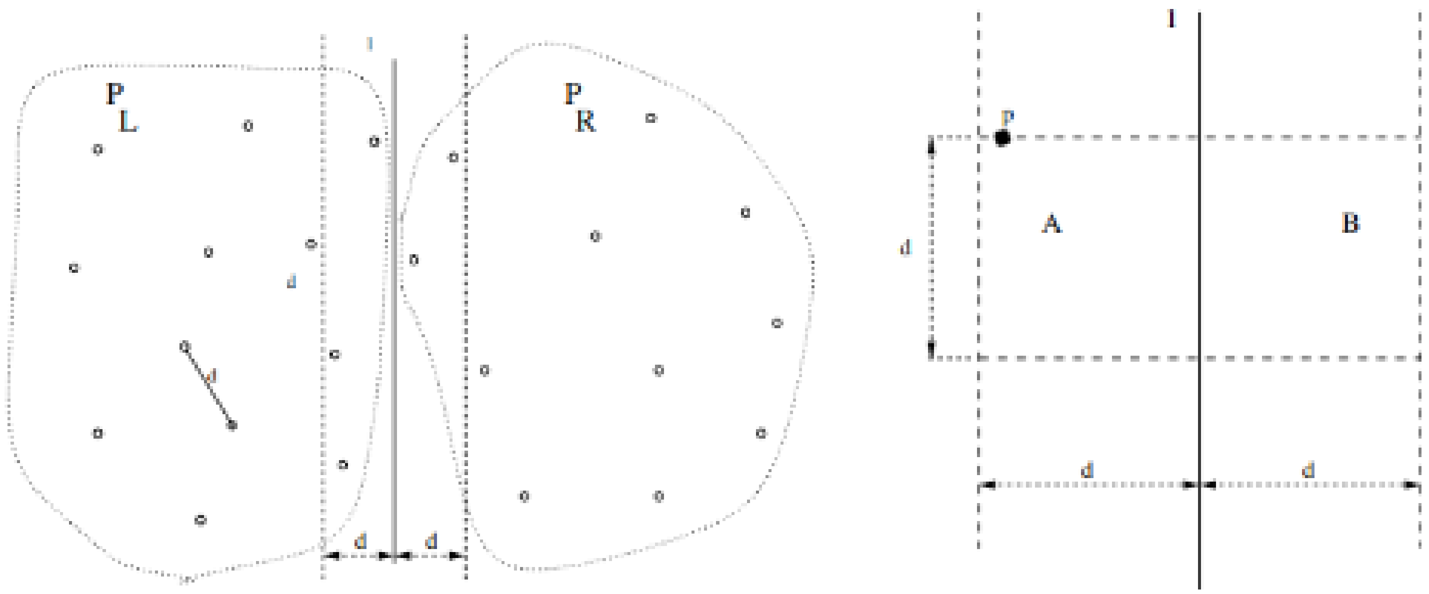
   Punkty znajdujące się na prostej l (o ile są takie) kwalifikujemy do tych podzbiorów w dowolny sposób.

2. { rekurencyjnie }
   $(i_1, j_1) \leftarrow$ para punktów z P_L o najmniejszej odległości;
   $(i_2, j_2) \leftarrow$ para punktów z P_R o najmniejszej odległości.

3. Niech (i', j') będzie tą parą punktów znalezioną w kroku 2, dla której odległość (oznaczmy ją przez d) jest mniejsza.
   Sprawdzamy czy istnieje para punktów (t, s) odległych o mniej niż d takich, że $t \in P_L$ i $s \in P_R$. Jeśli istnieje, przekazujemy ją jako wynik procedury, w przeciwnym razie jako wynik przekazujemy parę (i', j').

□

Wyjaśnienia wymaga sposób realizacji kroku 3. Oznaczmy przez P_C zbiór tych punktów z P, które leżą w odległości nie większej niż d od prostej l. Niech Y' oznacza tablicę Y, z której usunięto wszystkie punkty spoza P_C. Korzystamy z następującego spostrzeżenia:



Rysunek 4: *(a) W kroku 3 pary (t, s) należy szukać tylko w zaznaczonym pasie (b) Jeśli p ma być jednym z punktów pary (t, s), to drugi punkt musi znajdować się w kwadracie A. Ponadto wszystkie punkty z Y' leżące między t a s muszą leżeć w A lub w B.*

Fakt 3 *Jeśli (t, s) jest parą punktów odległych o mniej niż d taką, że $t \in P_L$ i $s \in P_R$, to t i s należą do P_C. Ponadto w tablicy w Y' pomiędzy t a s leży nie więcej niż 6 punktów.*

DOWÓD: Gdyby jeden z punktów leżał w odległości większej niż d od prostej l, to odległość między nimi byłaby większa niż d. Oczywiste jest też, że współrzędne y-kowe tych punktów różnią się nie więcej niż o d. Tak więc punkty t i s leżą w prostokącie o wymiarach $d \times 2d$ jak pokazano na rysunku 4.

W części A leżą tylko punkty z P_L. Ponieważ każde dwa z nich odległe są od siebie o co najmniej d, więc może ich tam znajdować się co najwyżej 4. Z analogicznego powodu w części B może znajdować się nie więcej niż 4 punkty z P_R. Tak więc w całym prostokącie znajduje się nie więcej niż 8 punktów.

□

Krok 3 sprowadza się więc do utworzenia tablicy Y', a następnie do obliczenia odległości każdego punktu z Y' do co najwyżej siedmiu punktów następujących po nim w tej tablicy.

3.5.2 Koszt:

Krok 1:

- Sortowanie - $\Theta(n \log n)$.
- Znalezienie prostej l i podział P na podzbiory - koszt stały.

Krok 2: $2T(n/2)$

Krok 3:

- Utworzenie Y' - $\Theta(n)$.
- Szukanie pary (t, s) - $\Theta(n)$.

Stąd koszt całego algorytmu wyraża się równaniem $T(n) = 2T(n/2) + \Theta(n \log n)$, którego rozwiązaniem jest $\Theta(n \log^2 n)$. Koszt ten można zredukować do $\Theta(n \log n)$. Wystarczy zauważyć , że sortowanie punktów w każdym wywołaniu rekurencyjnym jest zbyteczne. Zbiór P możemy przekazywać kolejnemu wywołaniu rekurencyjnemu jako tablice X i Y. Na ich podstawie można w czasie liniowym utworzyć odpowiednie tablice dla zbiorów P_L i P_R. Tak więc sortowanie wystarczy przeprowadzić jeden raz - przed pierwszym wywołaniem procedury rekurencyjnej.

Po takiej modyfikacji czas wykonania procedury rekurencyjnej wyraża się równaniem $T(n) = 2T(n/2) + \Theta(n)$, którego rozwiązaniem jest $\Theta(n \log n)$. Dodany do tego czas sortowania nie zwiększa rzędu funkcji.

ZADANIA EGZAMINACYJNE

Treść

Podać (jak najdokładniejsze) asymptotyczne ograniczenie na głębokość sieci przełączników realizujących wszystkie przesunięcia cykliczne ciągu wejściowego   Odp: 2log(n) - 1

Treść   **BEZ ROZWA**

Przymierzasz się do rozwiązania pewnego problemu i celujesz w algorytm działający w czasie $\Theta(\log\log n)$. Rozważasz zastosowanie metody dziel i zwyciężaj, więc złożoność twojego algorytmu będzie się wyrażać zależnością rekurencyjną

- $T(n) = O(1)$ dla n<const
- $T(n) = aT(f(n)) + g(n)$ w p.p.

Podaj jakie wartości może przyjąć stała a, oraz jakie mogą być funkcje f i g?

Treść   **BEZ ROZWA**

Rozwiąż równanie rekurencyjne (z redukcją do pierwiastka):

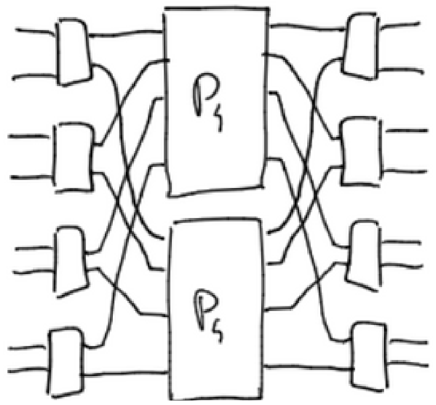
$$T(n) = \begin{cases} 1 : & n = 1 \\ T(\sqrt{n}) + O(1) : & \text{wpp.} \end{cases}$$

Możesz ograniczyć się do rozwiązania dla n mających odpowiednią postać (taką, by w trakcie redukcji argumenty dla T były liczbami naturalnymi).

Treść

Narysuj sieć Benesa-Waksmana dla n = 8.

▼ Rozwiązanie



Treść

Przypomnij sobie algorytm oparty na zasadzie Dziel i Zwyciężaj dla problemu znajdowania najbliższej pary punktów na płaszczyźnie.

Opisz trzecią fazę algorytmu, a więc tę, która następuje po wywołaniach rekurencyjnych.

Jaka jest jej złożoność?

▼ Rozwiązanie

Dla każdego punktu podziału dla współrzędnych x robimy odpowiedni padding(na podstawie aktualnej najkrótszej odległości między punktami, które oznaczamy `d`). Następnie przechodzimy po współrzędnych y, w określonym prostokącie wyznacznowym przez padding, z góry na dół i dodajemy do zbioru `Y` kolejne punkty. Po dodaniu nowego punktu sprawdzamy czy w zbiorze są takie punkty, których odległość do tego nowego jest większa niż `d`. Jeśli odległość jest większa niż `d` to usuwamy taki element ze zbioru. Jeśli jest więcej element w `Y` niż 6 to także usuwamy nadmiar punktów najbardziej odległych od nowego elementu. Po usunięciu zbyt odległych punktów przechodzimy do sprawdzenia czy odległość między którymś z elementem zbioru i nowym elementem jest lepsza niż `d`. Jeśli jest lepsza to ta wartość staje się nowym `d`. W taki sposób powtarzamy aż zejdziemy na sam dół. Następnie przechodzimy do następnego punktu podziału x i powtarzamy algorytm.

$\Theta(n)$

Zadanie 11. Rozwiąż poniższe równanie rekurencyjne:

$$T(n) = \begin{cases} 1 & \text{dla } n = 2 \\ 4T(\sqrt{n}) + \mathcal{O}(1) & \text{dla } n > 1 \end{cases}$$

Rozwiązanie (K. Kleczkowski). Niech $n > 1$.

Należy podstawić $n = 2^k$, by pozbyć się pierwiastka, przy czym $\mathbb{N} \ni k \geqslant 0$. Stąd otrzymujemy, że $T(2^k) = 4T(2^{k/2}) + \mathcal{O}(1)$. Podstawmy $U(k) = T(2^k)$. Mamy więc $U(k) = 4U(k/2) + \mathcal{O}(1) = 4U(k/2) + \mathcal{O}(k^0)$.

Ponieważ $\log_2(4) = 2 > 0$, innymi słowy, rozgałęzianie kosztuje więcej niż scalanie, to z tw. o rekursji uniwersalnej $U(k) = \mathcal{O}(k^{\log_2 4}) = \mathcal{O}(k^2)$; stąd że $T(n) = U(k)$, to $T(n) = \mathcal{O}(\log^2 n)$.

Treść

Podaj optymalny (pod względem liczby wykonywanych porównań) algorytm jednoczesnego znajdowania minimum i maksimum w ciągu.

Ile porównań wykonuje ten algorytm?

▼ Rozwiązanie

Rozwiązanie (Łyskawa). Porównujemy ze sobą parami kolejne elementy zbioru. Większy z nich przenosim do zbioru $S_{\max}$, większy do zbioru $S_{\min}$. $n/2$ porównań.

Wyszukujemy element minimalny zbioru $S_{\min}$ i element maksymalny zbioru $S_{\max}$, każde wyszukiwani to kolejnych $n/2 - 1$ porównań. Razem $3n/2 - 2$ porównań.

Treść

Jaką złożoność ma uogólnienie algorytmu Karatsuby, w którym mnożone liczby dzielone są na trzy części?

Odpowiedź uzasadnij.

▼ Rozwiązanie

Szkic dowodu.

Niech
$x = x1 + +x2 + +x3$
$y = y1 + +y2 + +y3$ (równej długości)
$x * y = x3 * y3 + \ldots$ (dużo rzeczy).
W takiej postaci mamy 9 mnożeń i to za dużo. W Karatsubie zmniejszamy liczbę mnożeń przedstawiając sumy $E(x_i * y_i)$ jakoś sprytnie (zamiast tego mamy więcej dodawań/odejmowań), ale nie pamiętam jak i ile wtedy zostaje mnożeń. Natomiast, zakładając, że 7:
$T(n) = 7 * T(n/3) + O(n)$
I wtedy
$T(n) = n^{(}log_3 7)$.

ZADANIA NA ĆWICZENIACH

1. (0,5pkt) Rozwiąż z dokładnością do Θ następującą rekurencję:

$$T(n) = \begin{cases} 1 & \text{dla } n = 1 \\ 2T(n/2) + n/\log n & \text{dla } n > 1 \end{cases}$$

2. (2pkt) Danych jest n prostych $l_1, l_2, \ldots l_n$ na płaszczyźnie ($l_i = a_i x + b_i$), takich że żadne trzy proste nie przecinają się w jednym punkcie. Mówimy, że prosta l_i jest widoczna z punktu p jeśli istnieje punkt q na prostej l_i, taki że odcinek $\overline{pq}$ nie ma wspólnych punktów z żadną inną prostą l_j ($j \neq i$) poza (być może) punktami p i q.

   Ułóż algorytm znajdujący wszystkie proste widoczne z punktu $(0, +\infty)$.

3. (2pkt) Zaproponuj modyfikację algorytmu Karatsuby, która oblicza kwadrat danej liczby. Rozważ podział liczby na k części:
   - dla $k = 2$,
   - dla $k = 3$.

   Postaraj się, by stałe używane przez algorytm były jak najmniejsze.
   Czy dla ustalonego k można otrzymać algorytm podnoszenia do kwadratu, który jest asymptotycznie szybszy od algorytmu mnożenia?

4. (1,5pkt) *Otoczką wypukłą* zbioru P, punktów na płaszczyźnie, nazywamy najmniejszy wielokąt wypukły zawierający (w swoim wnętrzu lub na brzegu) wszystkie punkty z P. Naturalny, oparty na zasadzie dziel i zwyciężaj, algorytm znajdowania otoczki wypukłej dla zbioru P, dzieli P na dwa (prawie) równoliczne podzbiory (np. "pionową" prostą), znajduje rekurencyjnie otoczki wypukłe dla tych podzbiorów, a następnie scala te otoczki. Podaj algorytm wykonujący tę ostatnią fazę algorytmu, tj. algorytm scalania dwóch otoczek wypukłych.

5. (1,5pkt) Dane jest drzewo binarne (możesz założyć dla prostoty, że jest to pełne drzewo binarne), którego każdy wierzchołek v_i skrywa pewną liczbę rzeczywistą x_i. Zakładamy, że wartości skrywane w wierzchołkach są różne. Mówimy, że wierzchołek v jest minimum lokalnym, jeśli wartość skrywana w nim jest mniejsza od wartości skrywanych w jego sąsiadach.

   Ułóż algorytm znajdujący lokalne minimum odkrywając jak najmniej skrywanych wartości.

6. Dane jest nieukorzenione drzewo z naturalnymi wagami na krawędziach oraz liczba naturalna C.

   (a) (2pkt) Ułóż algorytm obliczający, ile jest par wierzchołków odległych od siebie o C.

   (b) (Z 2,5pkt) Jak w punkcie (a), ale algorytm ma działać w czasie $O(n \log n)$.

   Uwaga: Można zadeklarować tylko jeden z punktów (a), (b).

7. (1pkt) *Inwersją* w ciągu $A = a_1, \ldots, a_n$ nazywamy parę indeksów $1 \leq i < j \leq n$, taką że $a_i > a_j$. Pokaż jak można obliczyć liczbę inwersji w A podczas sortowania przez scalanie.

8. (2pkt) Niech P_n będzie zbiorem przesunięć cyklicznych ciągu n-elementowego o potęgi liczby 2 nie większe od n. Pokaż konstrukcję sieci przełączników realizujących przesunięcia ze zbioru P_n.

   Uwagi:
   - Możesz założyć, że n jest potęgą dwójki albo szczególną potęgą dwójki, albo ...
   - Sieć Beneša-Waksmana jest dobrym rozwiązaniem wartym 0pkt (tzn. nic niewartym).

9. (Z 2pkt) *Dekompozycją centroidową* nazywamy następujący proces: dla danego drzewa T na n wierzchołkach znajdź wierzchołek $u \in T$ taki, że każda spójna składowa $T \setminus \{u\}$ ma rozmiar co najwyżej $n/2$, a następnie powtórz rozumowanie w każdej z tych spójnych składowych (o ile zawierają więcej niż jeden wierzchołek). Taka dekompozycja może być w naturalny sposób reprezentowana jako drzewo T' na n wierzchołkach, którego korzeniem jest u. Naiwna implementacja powyższej procedury działa w czasie $O(n \log n)$. Skonstruuj algorytm, który konstruuje T' w czasie $O(n)$.

Zadania dodatkowe - nie będą rozwiązywane w czasie ćwiczeń

1. (1pkt) Ułóż, oparty o zasadę dziel i zwyciężaj, algorytm obliczający największy wspólny dzielnik dwóch liczb, który wykorzystuje następującą własność:

$$gcd(a,b) = \begin{cases} 2gcd(a/2, b/2) & \text{gdy } a, b \text{ są parzyste,} \\ gcd(a, b/2) & \text{gdy } a \text{ jest nieparzyste a } b \text{ jest parzyste,} \\ gcd((a-b)/2, b) & \text{gdy } a, b \text{ są nieparzyste} \end{cases}$$

□

   Porównaj złożoność tego algorytmu z algorytmem Euklidesa.

2. (Z 1,5pkt) Algorytm Euklidesa wyznacza $\gcd(x, y)$ w czasie $O(\log \min(x, y))$. Skonstruuj algorytm, który wyznacza $\gcd(x_1, \ldots, x_n)$ w czasie $O(n + \log \min(x_1, \ldots, x_n))$.

3. (1,5pkt) Macierz A rozmiaru $n \times n$ nazywamy macierzą Toeplitza, jeśli jej elementy spełniają równanie $A[i,j] = A[i-1, j-1]$ dla $2 \leq i, j \leq n$.

   (a) Podaj reprezentację macierzy Toeplitza, pozwalającą dodawać dwie takie macierze w czasie $O(n)$.

   (b) Podaj algorytm, oparty na metodzie "dziel i zwyciężaj", mnożenia macierzy Toeplitza przez wektor. Ile operacji arytmetycznych wymaga takie mnożenie?

4. (Z 2pkt) *Dekompozycją centroidową* nazywamy następujący proces: dla danego drzewa T na n wierzchołkach znajdź wierzchołek $u \in T$ taki, że każda spójna składowa $T \setminus \{u\}$ ma rozmiar co najwyżej $n/2$, a następnie powtórz rozumowanie w każdej z tych spójnych składowych (o ile zawierają więcej niż jeden wierzchołek). Taka dekompozycja może być w naturalny sposób reprezentowana jako drzewo T' na n wierzchołkach, którego korzeniem jest u. Naiwna implementacja powyższej procedury działa w czasie $O(n \log n)$. Skonstruuj algorytm, który konstruuje T' w czasie $O(n)$.

5. (2pkt) Jakie jest prawdopodobieństwo wygenerowania permutacji identycznościowej przez sieć Beneša-Waksmana, w której przełączniki ustawiane są losowo i niezależnie od siebie w jeden z dwóch stanów (każdy stan przełącznika jest osiągany z prawdopodobieństwem $1/2$).

6. (0 pkt) Przypomnij sobie algorytm scalający dwie posortowane tablice U i V w czasie liniowym, tj. w czasie liniowo proporcjonalnym do sumy długości tych tablic.

7. (1 pkt) Złożoność podanego na wykładzie algorytmu sortowania przez scalenia wyraża się wzorem:

$$T(1) = a$$
$$T(n) \leq T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + bn \qquad \text{dla } n > 1$$

   dla pewnych $a, b > 0$. Udowodnij, że $T(n) \in O(n \log n)$.

8. (1 pkt) Zamiast dzielić tablicę T na dwie połówki, algorytm sortowania przez scalanie mógłby dzielić ją na części o rozmiarach $\lceil n/3 \rceil$, $\lceil (n+1)/3 \rceil$ oraz $\lceil (n+2)/3 \rceil$, sortować niezależnie każdą z tych części, a następnie scalać je. Podaj bardziej formalny opis tego algorytmu i przeanalizuj czas jego działania.

9. (1pkt) Rozważ wersje algorytmu *multiply* dzielące czynniki na trzy i cztery części. Oblicz współczynniki w kombinacjach liniowych określających wartości c_i.

10. (1 pkt) Niech u i v będą liczbami o n i m cyfrach (odpowiednio). Załóżmy, że $m \leq n$. Klasyczny algorytm oblicza iloczyn tych liczb w czasie $O(mn)$. Algorytm *multiply* z wykładu potrzebuje $O(n^{\log 3})$ czasu, co jest nie do zaakceptowania gdy m jest znacznie mniejsze od n. Pokaż, że w takim przypadku można pomnożyć liczby u i v w czasie $O(nm^{\log(3/2)})$.

11. (1pkt) Udowodnij, że podana na wykladzie sieć przełączników (sieć Beneša-Waksmana) jest asymptotycznie optymalna pod względem głębokości i liczby przełączników.

12. (1pkt) Załóżmy, że przełączniki ustawiane są losowo (każdy przełącznik z jednakowym prawdopodobieństwem ustawiany jest w jeden z dwóch stanów). Sieć zbudowaną z takich przełączników można traktować jako generator losowych permutacji. Udowodnij, że nie istnieje sieć przełączników, generująca permutacje z rozkładem jednostajnym.

13. (2pkt) Przeanalizuj sieć permutacyjną omawianą na wykładzie (tzw. sieć Beneša-Waksmana)
    - Pokaż, że ostatnią warstwę przełączników sieci Beneša-Waksmana można zastąpić inną warstwą, która zawiera $n/2 - 1$ przełączników (a więc o jeden mniej niż w sieci oryginalnej) a otrzymana sieć nadal będzie umożliwiać otrzymanie wszystkich permutacji.
    - Uogólnij sieć na dowolne n (niekoniecznie będące potęgą liczby 2).

14. *Medianą* n elementowego wielozbioru A nazywamy wartość tego elementu z A, który znalazłby się na pozycji $\lceil n/2 \rceil$ po uporządkowaniu A według porządku $\leq$. Ułóż algorytm, który dla danych uporządkowanych niemalejąco n-elementowych tablic $T_1, T_2, \ldots, T_k$ znajduje medianę wielozbioru A utworzonego ze wszystkich elementów tych tablic.

    (a) (1,8pkt) Rozwiąż to zadanie dla $k = 3$.

    (b) (Z)(2pkt) Rozwiąż to zadanie dla dowolnej stałej $k > 3$.

15. (2pkt) Przeanalizuj następujący algorytm oparty na strategii dziel i zwyciężaj jednoczesnego znajdowania maksimum i minimum w zbiorze $S = \{a_1, \ldots, a_n\}$:

```
Procedure MaxMin(S:set)
  if |S|= 1 then return {a1, a1}
  else
    if |S|= 2 then return (max(a1, a2), min(a1, a2))
    else
      podziel S na dwa równoliczne (z dokładnością do jednego elementu) podzbiory S1, S2
      (max1, min1) ← MaxMin(S1)
      (max2, min2) ← MaxMin(S2)
      return (max(max1, max2), min(min1, min2))
```

Uwaga: Operacja **return** $(\max(a_1, a_2), \min(a_1, a_2))$ wykonuje jedno porównanie.

- Jak pokazaliśmy na jednym z wykładów każdy algorytm dla tego problemu, który na elementach zbioru wykonuje jedynie operacje porównania, musi wykonać co najmniej $\lceil \frac{3}{2}n - 2 \rceil$ porównania. Dla jakich danych powyższy algorytm wykonuje tyle porównań? Podaj wzorem wszystkie takie wartości.
- Jak bardzo może różnić się liczba porównań wykonywanych przez algorytm od dolnej granicy?
- Popraw algorytm, tak by osiągał on tę granicę dla każdej wartości n?