Zadanie 6.

Niech $\{k_1, k_2\} = DualPivotPartition(A, p, q)$ będzie procedurą dzielącą tablicę $A[p...q]$ na trzy pod-tablice: $A[p...k_1-1]$, $A[k_1+1...k_2-1]$, $A[k_2+1...q]$, wykorzystującą losowe 2 elemeny jako pivoty. Podaj pseudokod algorytmu $DualPivotRandomSelect()$ znajdujący i-tą statystykę pozycyjną w tablicy A. Algorytm niech wykorzystuje procedurę $DualPivotPartition()$. Zapisz wzór rekurencyjny na wartość oczekiwaną liczby porównań w stworzonym algorytmie.

Rozpiszmy wstępnie pseudo-kod algorytmu $DualPivotRandomSelect(A, p, r, i)$

A - tablica
p - index początkowy tab
r - index końcowy tab
i - szukana statystyka pozycyjna
lp - index lewego pivota
rp- index prawego pivota

Algorithm 1 DualPivotRandomSelect(A,p,r,i)

if p = r **then**
    then return A[p]
end if
//rp i lp zostanie zwrócony z procedury
//rp zostanie przypisany g z DPP i lp zostanie przypisany j z DPP
(lp,rp) = DualPivotPartition(A,p,r)
$k_1 = (lp - p) + 1$
$k_2 = (rp - p) + 1$
if $i \leqslant k_1$ **then**
    then return DualPivotRandomSelect(A,p,lp,i)
else if $k_1 \leqslant i \leqslant k_2$ **then**
    then return DualPivotRandomSelect(A,lp+1,rp,$i - k_1$)
else
    return DualPivotRandomSelect(A,rp+1,r,$i - k_2$)
end if

Widzimy już zatem schemat działania głównego algorytmu, w zależności od podziału tablicy przez $DualPivotPartition()$, algorytm wywouje sam siebie rekurencyjnie, do momentu odnalezienia i-tej statystyki pozycyjnej.

Zajmijmy się teraz procedurą $DualPivotPartition()$
Oznaczenia:
A - tablica
low - początkowy index
high - końcowy index
lp - left pivot przekazany przez główny algorytm
x - bufor dla left pivot
y - bufor dla right pivot

Algorithm 2 DualPivotPartition(A,low,high)

```
(x, y) = getRandomPair()
//zwraca pare losowych elementów x ¡ y
//aby algorytm dzialal poprawnie
//arr[low]=x i arr[high]=y
//dlatego tez zamieniamy elementy tablicy, ktore sa rowne x i y z arr[low] i arr[high]
//jest to pseudokod, dlatego tez nie jest tu istotny sam proces poszukiwania indexów x i y, ale idea zamiany
swap(arr[low],arr[Xindex])
swap(arr[high],arr[Yindex])
j = low + 1
g = high - 1
k = low + 1
while k ⩽ g do
    if A[k] ⩽ x then
        swap(A[k], A[j])
        j = j + 1
    else if A[k] ⩾ y then
        while A[g] > y&&k < g do
            g = g − 1
        end while
        swap(A[k], A[g])
        g = g − 1
        if A[k] < x then
            swap(A[k], A[j])
            j = j + 1
        end if
    end if
    k = k + 1
end while
j = j − 1
g = g + 1
swap(A[low], A[j])
swap(A[high], A[g])
return j,g
```

Korzystając z pivotów, procedura porządkuje elementy tablicy, przezucając elementy mniejsze od lewego pivota, przed niego i analogicznie po-

Zajmiemy się teraz wyznaczeniem górnego ograniczenia $T(n)$ na oczekiwany czas działania procedury RANDOMIZED-SELECT dla tablicy o n elementach. W podrozdziale 8.4 zauważyliśmy, że algorytm RANDOMIZED-PARTITION wyznacza podział, którego dolna część ma 1 element z prawdopodobieństwem $2/n$, a i elementów z prawdopodobieństwem $1/n$, dla $i = 2, 3, \ldots, n-1$. Zakładając, że funkcja $T(n)$ jest monotonicznie rosnąca, w najgorszym przypadku w procedurze RANDOMIZED-SELECT i-ty element zawsze należy wyznaczyć w większej części podziału. Mamy zatem następującą zależność rekurencyjną:

$$T(n) \leqslant \frac{1}{n}\left(T(\max(1, n-1)) + \sum_{k=1}^{n-1} T(\max(k, n-k))\right) + O(n)$$

223

stępując z pivotem prawym, następnie pivoty są dalej przekazywane wraz z upoirządkowaną tablicą do algorytmu głównego.

Rozpatrzmy równanie rekurencyjne na ilość porównań. Dla uproszczenia analizy, zakładamy, że elementy wtablicy są różne. Założymy, że jako pivoty wybieramy z jednakowym prawdopodobieństwem p_{k_1,k_2} każdą z $\binom{n}{2}$ posortowanych par liczb: $p_{k_1,k_2} = 1/\binom{n}{k} = \frac{2}{n(n-1)}$. Skoro rozpatrywane jest ograniczenie górne, należy wybierać największą z 3 rozdzielonych podtablic. Dlatego też bazując na równaniu opisanym w książce Wstęp do algorytmów, autorstwa Cormena:

$$T(n) \leqslant \sum_{1\leqslant k1<k2<n}(p_{k1,k2}T(max(k1-1,k2-1-(k1+1)+1,n-k2))+O(n)$$