Algorytmy i struktury danych
Lista 6

Zadanie 2.

```
function longest_sequence(X, N)

     P = array of length N
     M = array of length N + 1

     L = 0
     for i in range 0 to N-1:                                  (I)
          low = 1
          high = L
          while low  <= high:                                  (II)
               mid = ceil((low+high)/2)
                if X[M[mid]] < X[i]:
                       low = mid+1
               else:
                       high = mid-1

          newL = low
          P[i] = M[newL-1]
          M[newL] = i

          if newL > L:
                L = newL


     S = array of length L
     k = M[L]
     for i in range L-1 to 0:                                 (III)
           S[i] = X[k]
            k = P[k]
     return S
```

Powyższy algorytm rozwiązuje problem znalezienia najdłuższego ciągu rosnącego o długości N, za pomocą prostych operacji na tablicach oraz binary searcha. W głównej pętli (I) odbywa się badanie poszczególnych elementów ciągu X. Tablice P i M służą do zapisywania wyników:$M[j]$ przechowuje indeksy wartości ($X[k]$), kończących najdłuższe ciągi o długości j, dla $k <= i$ (gdzie i to badany indeks X, w poszczególnych iteracjach (I)), a P to poprzednik $X[k]$ w znalezionym najdłuższym rosnącym ciągu. L jest wartością pomocniczą przechowującą długość najlepszego znalezionego ciągu.

Pętla (II) szuka największego $j <= L$ (low jest odpowiednikiem j), na podstawie którego będzie powstawał poszukiwany ciąg. Binary search został wykorzystany na tablicy M, ponieważ $X[M[0]], X[M[1]] \ldots$ jest ciągiem rosnącym. W przypadku gdy sprawdzany element $X[i]$ jest większy od $X[M[mid]]$ "zostawiamy" go w tablicy M, w przeciwnym wypadku kontynuujemy sprawdzanie pierwszej połowy tablicy M (odrzucając kolejno poszczególne elementy M). Po wykonaniu pętli (II) następuje aktualizacja elementów M i P oraz L (w przypadku, gdy znaleziony ciąg okazał się najdłuższy).

Po zakończeniu działania głównej części algorytmu, do tablicy S zostaje zapisany najdłuższy znaleziony ciąg rosnący (III).

Powyższy algorytm posiada złożoność $O(nlogn)$. Pętla w (I) wykona się w najgorszym razie n - 1 razy, więc jej złożoność można oszacować jako $O(n)$ Pętla w (II) to binary search, który jak wiemy w najgorszym przypadku posiada złożoność $O(logn)$. Pętla (III) w najgorszym przypadku będzie się składała z n - 1 elementów. Co ostatecznie nam daje

$$O(nlogn) + O(n) = O(nlogn)$$