

# Algorytmy i struktury danych

## Lista zadań 2

### Zadanie 1

Ile trzeba porównań, by znaleźć element  $x$  w nieuporządkowanej tablicy  $\mathbf{t}$  o rozmiarze  $n$ . Oblicz wartość średnią i wariancję zakładając, że element  $x$  może znajdować się z jednakowym prawdopodobieństwem, pod dowolnym indeksem tablicy.

$$E(X) = \sum_{i=1}^n x_i p_i = \sum_{i=1}^n i \cdot \frac{1}{n} = \frac{\left(\frac{1}{n} + \frac{n}{n}\right) n}{2} = \frac{n+1}{2}$$

$$Var(X) = E(X^2) - E(X)^2 = \frac{n^2+1}{2} - \left(\frac{n+1}{2}\right)^2 = \frac{2n^2+2}{4} - \frac{n^2+2n+1}{4} = \frac{(n-1)^2}{4}$$

### Zadanie 2

Bisekcja. Ile trzeba porównań, by znaleźć element  $x$  w posortowanej tablicy  $\mathbf{t}$  o rozmiarze  $n$ . Podaj minimalną wartość gwarantującą sukces i strategię, jak to zrobić. Postaraj się podać wzór ogólny, który pozwoli wyliczyć dokładną wartość dla dowolnego  $n$ . Sprawdź go dla  $n = 1, \dots, 20$ .

1. Oblicz środek przedziału.
2. Jeżeli wartość w środku przedziału jest większa od  $x$ , to środek staje się lewym końcem przedziału.
3. W przeciwnym wypadku środek staje się prawym końcem przedziału.

$$n = 1 \implies 1$$

$$n = 2 \implies 2$$

$$n = 4 \implies 3$$

$$n = 8 \implies 4$$

$$n = 16 \implies 5$$

$$n = 20 \implies 5$$

$$\lfloor \log_2(n) \rfloor + 1$$

### Zadanie 3

Rozważ trzy wersje znajdowania maksimum w tablicy `int maks(int t[], int n)`. Ile porównań między elementami tablicy  $n$ -elementowej wykonuje każda z wersji? Ile pamięci wymaga każda z tych wersji? Uwzględnij fakt, że głębokość rekurencji ma wpływ na zużycie pamięci, ponieważ powstaje wiele kopii zmiennych lokalnych. Która wersja jest więc najlepsza?

(a) iteracyjna: `{int x = a[--n]; while(n-->0) if(t[n] > x) x = t[n]; return x;}`

$n - 1$  porównań i 3 zmienne lokalne

(b) rekurencyjnie oblicza maksimum  $n - 1$  elementów i porównuje z ostatnim elementem

$n - 1$  porównań i  $2n$  zmiennych lokalnych

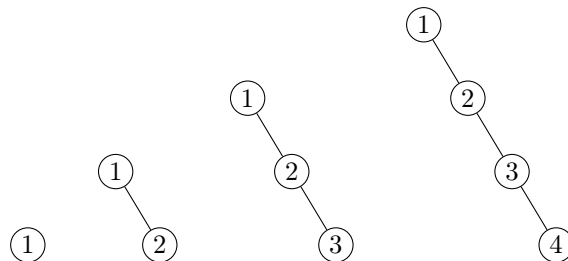
(c) dzieli tablicę na dwie części, rekurencyjnie znajduje ich maksima i wybiera większe z nich.

$2(n - 1) + 1 = n - 1$  porównań i  $2(n - 1) + 2 = 2n$  zmiennych lokalnych

### Zadanie 4

Jakie drzewo powstanie po wstawieniu do pustego drzewa BST liczb od 1 do  $n$  w kolejności rosnącej? Jaka potem będzie głębokość drzewa? Ile porównań kluczy wykonano w trakcie tworzenia tego drzewa? Jaka jest złożoność w tego procesu w notacji  $O$ ?

Uwaga: Element wstawiamy na pierwsze napotkane puste miejsce zaczynając od korzenia. Jeśli miejsce jest zajęte, to gdy element jest mniejszy od klucza w węźle, idziemy do lewego poddrzewa, a gdy większy lub równy – do prawego poddrzewa.



Głębokość drzewa:  $n$

$$\text{Ilość porównań: } \sum_{i=1}^n i - 1 = \frac{n(n-1)}{2} = \frac{n^2 - n}{2}$$

$$\text{Złożoność: } \frac{n^2 - n}{2} = O(n^2)$$

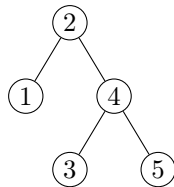
## Zadanie 5

Implementacja usuwania węzła  $X$  z drzewa binarnego działa wg następującego schematu:

- (a) jeśli  $X$  nie ma dzieci, to go usuwamy a wskaźnik na  $X$  zmieniamy na  $\text{NULL}$ .
- (b) jeśli  $X$  ma jedno dziecko  $Y$ , to usuwamy  $X$ , a wskaźnik na  $X$  zastępujemy wskaźnikiem na  $Y$ .
- (c) jeśli  $X$  ma dwoje dzieci, to znajdujemy najmniejszy element  $Y$  w jego prawym poddrzewie, dane i klucz z węzła  $Y$  kopiujemy do  $X$  i usuwamy  $Y$ .

Uzasadnij, dlaczego postępowanie wg punktu (c) nie psuje prawidłowego rosnącego porządku kluczy wypisywanych w porządku *inorder* i dlaczego  $Y$  ma co najwyżej jedno dziecko, więc do jego usunięcia można zastosować punkt (a) lub (b).

Zastąpienie węzła  $X$  mającego dwoje dzieci polegające na znalezieniu najmniejszego elementu w prawym poddrzewie  $X$  nie psuje porządku *inorder*, ponieważ najmniejszy element prawego poddrzewa jest idealnym kandydatem na jego miejsce, dzięki temu że jest większy od wszystkich elementów w lewym poddrzewie  $X$ , a mniejszy od wszystkich elementów w prawym poddrzewie  $X$ .



Do usunięcia węzła  $Y$  możemy zastosować punkt (a) lub (b), ponieważ  $Y$  ma co najwyżej jedno dziecko, ponieważ jeśli  $Y$  ma dwoje dzieci, to jego lewe dziecko było by mniejsze od  $Y$ , co oznacza że  $Y$  nie jest najmniejszym elementem.

## Zadanie 6

Uzasadnij, że w każdym drzewie BST zawsze ponad połowa wskaźników (pól *left* i *right*) jest równa  $\text{NULL}$ .

Na każdy węzeł drzewa BST przypadają dwa wskaźniki, *left* i *right*, które wskazują na lewe i prawe poddrzewo. Drzewo składające się z samego korzenia ma 2 wskaźniki równe  $\text{NULL}$ . Każde dodanie węzła zwiększy łączną liczbę węzłów o 2 i zmniejszy liczbę węzłów równych  $\text{NULL}$  o 1.

Liczba węzłów:  $n$

Liczba wskaźników:  $2n$

Liczba wskaźników równych  $\text{NULL}$ :  $n + 1$

$$n + 1 > \frac{2n}{2}$$

$$n + 1 > n$$

## Zadanie 7

Ile maksymalnie węzłów może mieć drzewo BST o głębokości  $h$ ? Wylicz dokładną wartość, przyjmując, że głębokość oznacza ilość poziomów, na których występują węzły (sam korzeń:  $h = 1$ , korzeń i dzieci:  $h = 2 \dots$ ). Skorzystaj z wzoru na sumę ciągu geometrycznego. Wynioskuj, jaka jest najmniejsza, a jaka największa głębokość drzewa binarnego o  $n$  węzłach?

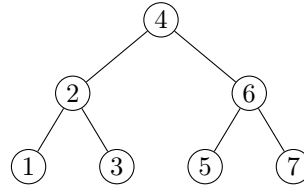
$$\text{Maksymalna ilość węzłów: } \frac{a_1(1 - q^n)}{1 - q} = \frac{1(1 - 2^h)}{1 - 2} = 2^h - 1$$

$$\text{Najmniejsza głębokość: } 1 + \lfloor \log_2(n) \rfloor$$

$$\text{Największa głębokość: } n$$

## Zadanie 8

Przeanalizuj operacje `find`, `insert`, `remove` zawarte w pliku `tree-2023-recursive.cc`. Jak ich pesymistyczna złożoność czasowa  $T(h)$  zależy od głębokości drzewa  $h$ ?



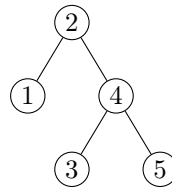
$$\text{find: } T(h) = \Theta(h)$$

$$\text{insert: } T(h) = \Theta(h)$$

$$\text{remove: } T(h) = \Theta(h)$$

## Zadanie 9

W pliku `tree-2023-recursive.cc` znajdziesz funkcję `int height(node *t)`, która wyliczy głębokość (ilość poziomów na jakich występują węzły) drzewa BST. Jak zależy czas wykonania tej funkcji od ilości  $n$  węzłów drzewa i/lub jego głębokości  $h$ ? To samo zadanie wykonaj też dla funkcji `int count(node *t)`.



height

$$T(n) =$$

$$T(h) =$$

count

$$T(n) =$$

$$T(h) =$$