Algorytmy i struktury danych

Zaawansowane algorytmy sortowania

Witold Marańda
maranda@dmcs.p.lodz.pl

1

Sortowanie za pomocą malejących przyrostów – metoda Shella

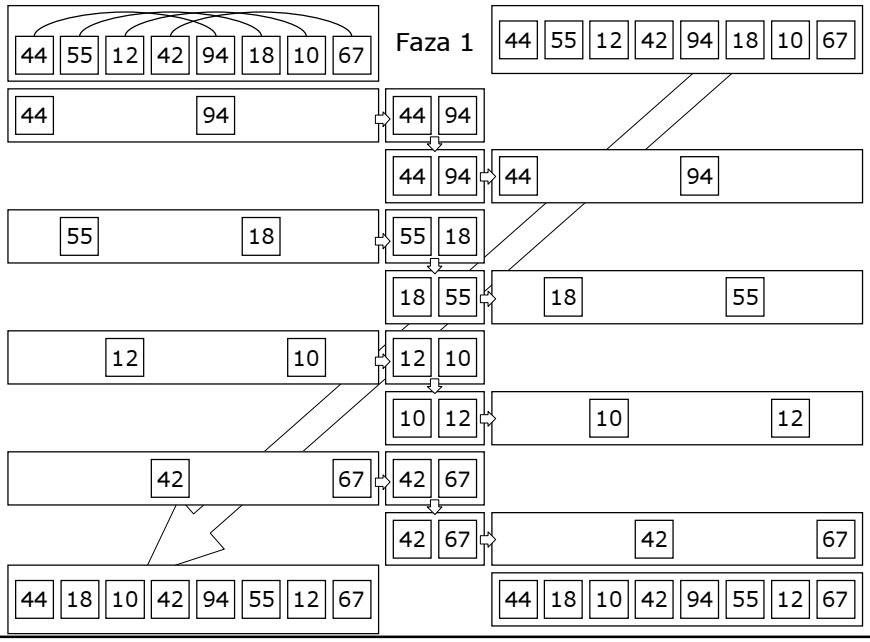
Metoda jest rozwinięciem metody sortowania przez wstawianie.

W metodzie tej, najpierw grupuje się i sortuje oddzielnie wszystkie elementy oddalone o pewną odległość (przyrost) *h* (tj. oddalone „co *h*”). W pierwszym kroku metody tworzy się więc *h*-podzbiorów, które sortowane są metodą przez wstawianie.

W następnych krokach powtarza się taką operację dla coraz mniejszych odległości *h*, aż do momentu gdy *h*=1, co odpowiada normalnemu sortowaniu całego zbioru (elementy oddalone „co jeden”).

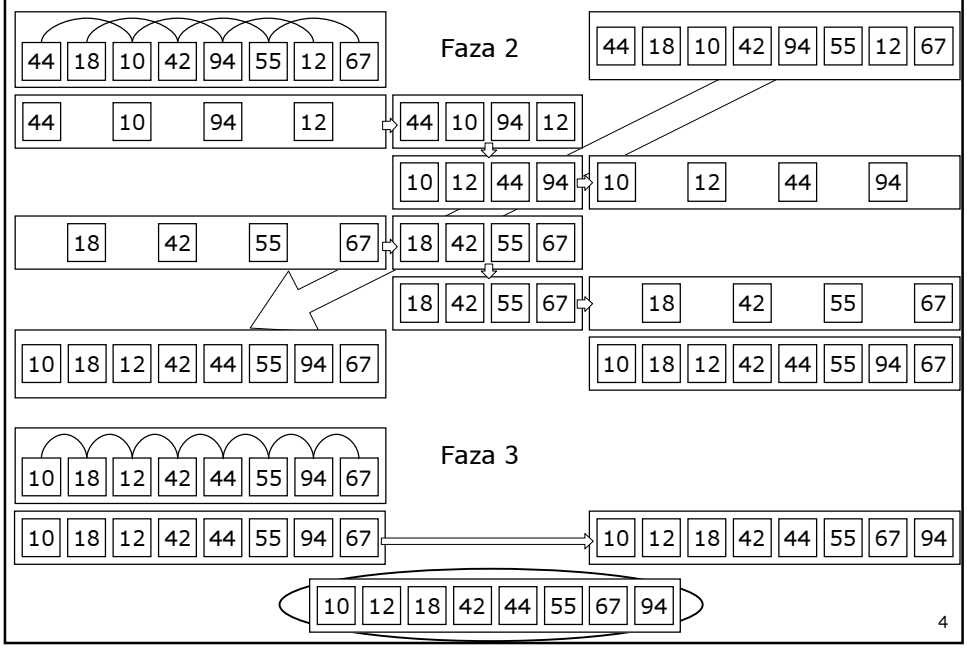
2

Sortowanie metodą Shella – przykład 1/2



3

Sortowanie metodą Shella – przykład 2/2



4

Sortowanie za pomocą malejących przyrostów – metoda Shella

Metoda ta jest bardzo efektywna, pomimo, że występuje kilka wstępnych procesów sortowań. Jednak dla dużych wartości odstępu *h*, sortowane zbiory mają mało elementów, a dla małych *h* zbiory są już znacznie uporządkowane i w obu tych przypadkach sortowanie takich zbiorów za pomocą metody przez wstawiane jest bardzo szybkie.

Metoda działa najlepiej gdy przyrosty *h* nie są swoimi dzielnikami. Zaleca się stosowanie następujących przyrostów:

$h_{k-1}=3h_k+1$, $h_t=1$, $t=\log_3 n-1$ (czyli: ... 121, 40, 13, 4, 1)

lub:

$h_{k-1}=2h_k+1$, $h_t=1$, $t=\log_2 n-1$ (czyli: ... 31, 15, 7, 3, 1)

Efektywność metody: Po, Pr ~ $\mathbf{n^{1,2}}$

5

Sortowanie przez podział (sortowanie szybkie)

W metodzie tej wybiera się losowo jakiś element *x* sortowanego zbioru. Następnie przegląda się zbiór od strony „lewej", aż znaleziony zostanie taki element A_i, że $A_i \geq x$, a od strony „prawej" znajduje się element A_j, że $A_j \leq x$. Następnie zamienia się miejscami elementy A_i i A_j i kontynuuje się proces przeglądania i zamiany, aż nastąpi spotkanie gdzieś w środku tablicy.

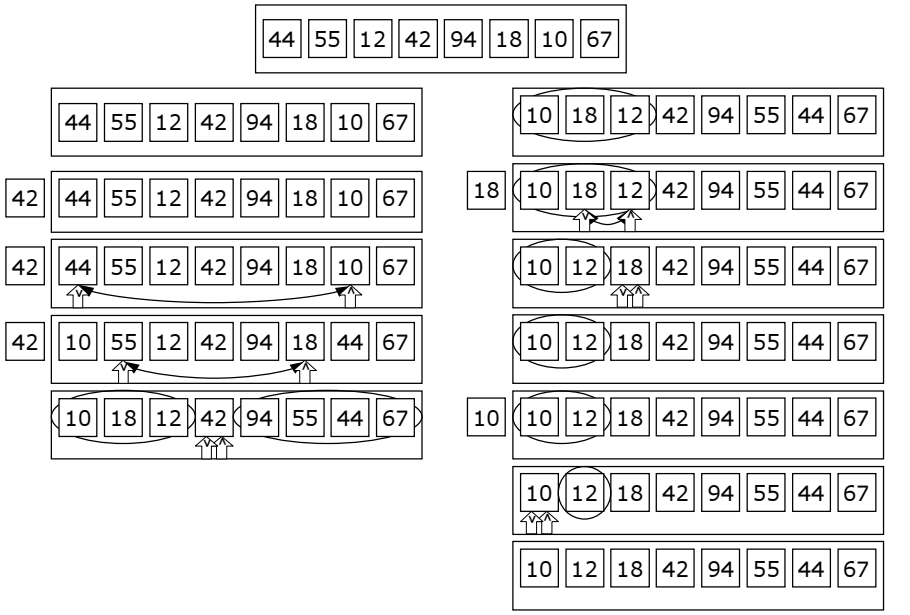
Miejsce spotkania wyznacza punkt podziału tablicy na dwie części. „Lewa" część składa się z elementów nie większych niż wybrany element *x,* „prawa" zaś z elementów nie mniejszych niż *x.*

Takie części sortuje się następnie w sposób jaki opisano powyżej. Powtarzanie tych operacji aż do momentu gdy części tablicy będą składały się z jednego elementu, doprowadzi do posortowania całej tablicy.

Efektywność metody: Po ~ n*log(n), Pr ~ **n**

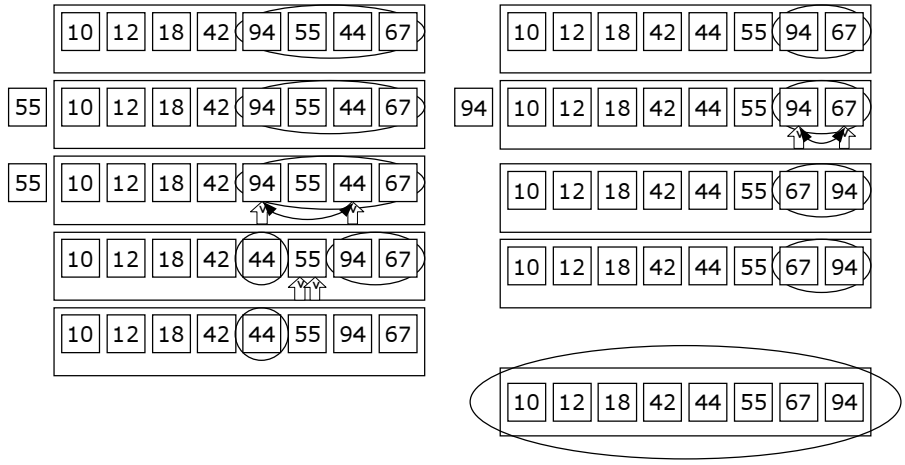
6

Sortowanie przez podział – przykład 1/2



7

Sortowanie przez podział – przykład 2/2



8

Sortowanie przez podział – program

```
PROCEDURE sort_szybkie(l, p: integer);
VAR
   i, j: integer;
   x, w: obiekt;
BEGIN
   i:=l;
   j:=p;
   x:=A[(l+p) div 2];
   REPEAT
      WHILE A[i].klucz < x.klucz DO i:=i+1;
      WHILE A[j].klucz > x.klucz DO j:=j-1;
      IF i<=j THEN
      BEGIN
         w:=A[i];
         A[i]:=A[j];
         A[j]:=w;
         i:=i+1;
         j:=j-1;
      END
   UNTIL i>j;
   IF l<j THEN sort_szybkie(l, j);
   IF i<p THEN sort_szybkie(i, p);
END;
```

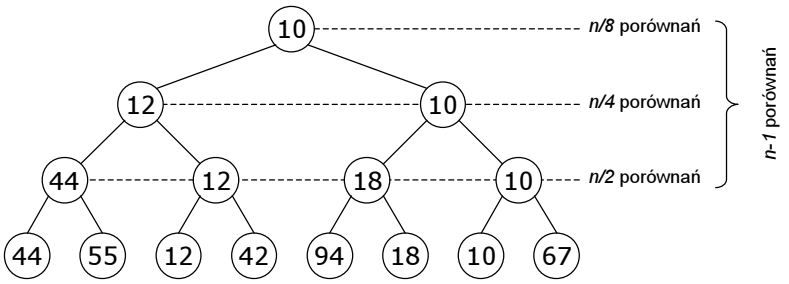
Początkowe wywołanie procedury:
`sort_szybkie(1, n);`

9

Sortowanie stogowe

10

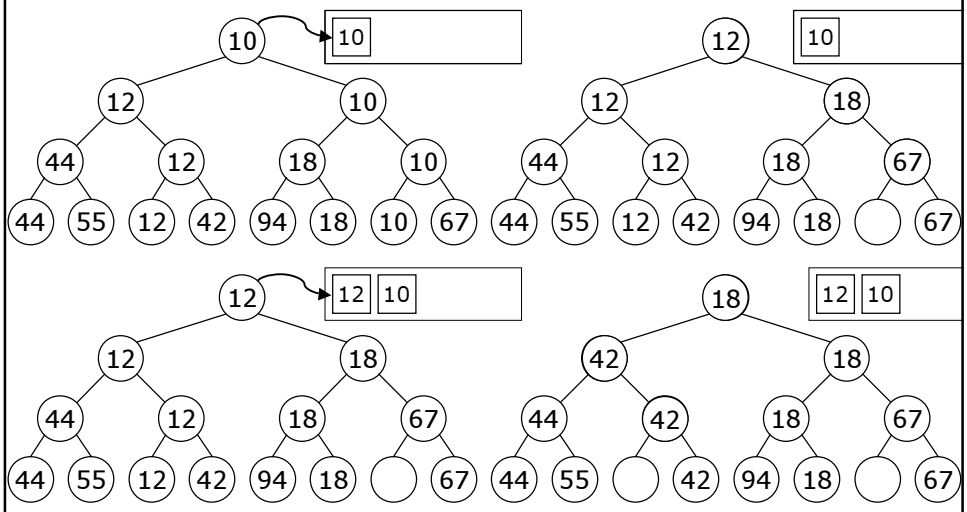
Drzewo porównań



- Dla *n*-elementowej tablicy można wyznaczyć „drzewo porównań” za pomocą *n-1* operacji porównań kluczy elementów
- Każdy węzeł jest elementem o mniejszym kluczu z dwóch sąsiadujących w drzewie
- Na wierzchołku drzewa zawsze znajduje się element o najmniejszym kluczu !

11

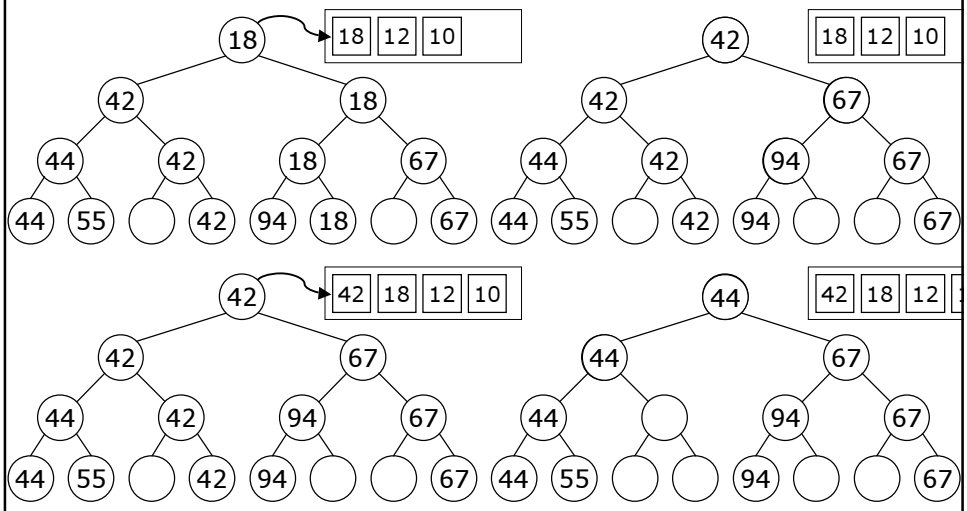
Wybieranie z drzewa



- Sortowanie tablicy, dla której utworzono drzewo wymaga:
  - pobrania elementu z wierzchołka (zawsze najmniejszy klucz)
  - zastąpienie pobranego elementu elementem o mniejszym kluczu z niższego węzła
- Procedura taka pozwala odczytać z drzewa posortowane elementy tablicy

12

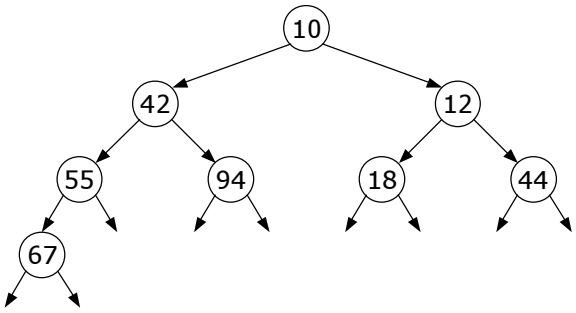
Sortowanie drzewiaste



- Otrzymanie posortowanej tablicy wymaga *n* operacji odczytu z drzewa
- Każdy odczyt (wybieranie elementu z drzewa wymaga $log_2(n)$ porównań i przesunięć
- Cały proces sortowania przez wybieranie z drzewa wymaga więc *n*log(n)* operacji (oraz *n-1* operacji potrzebnych do utworzenia drzewa)

13

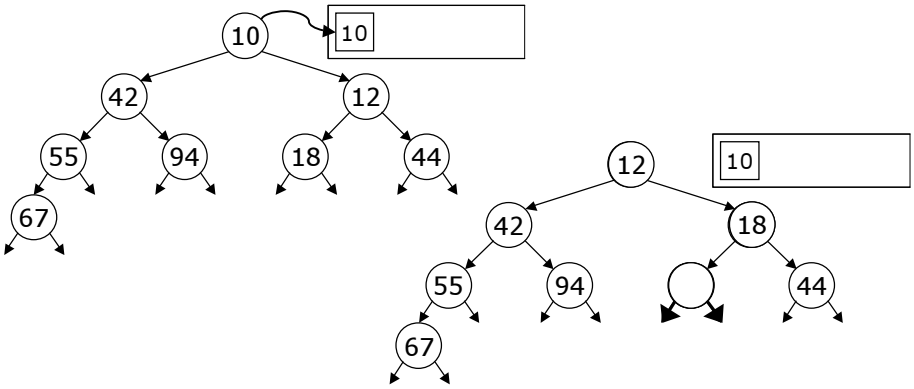
Stóg



- Stóg jest strukturą drzewiastą, której każdy element jest **nie większy** od dwóch elementów bezpośrednio pod nim (potomków)
- Pomiędzy elementami na tym samym poziomie nie zachodzą żadne relacje
- Tworzenie struktury stogu wymaga (jak poprzednio *n*log(n)* operacji

14

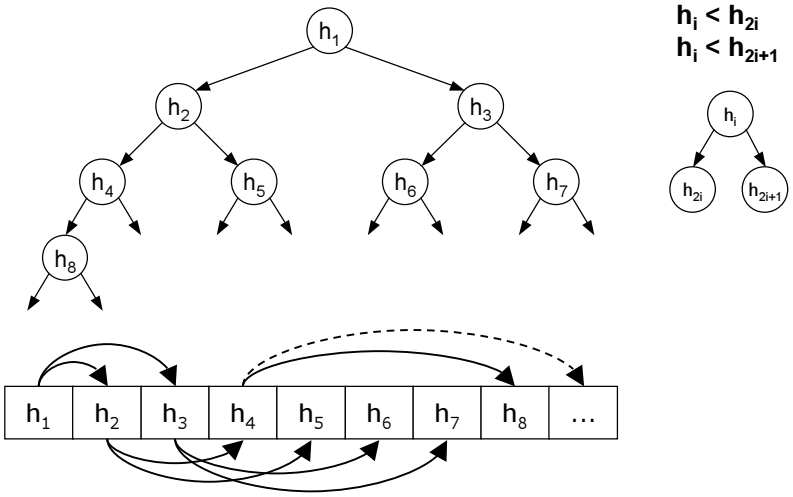
Sortowanie stogowe



- Sortowanie stogowe polega na pobraniu elementu z wierzchołka, przesuwaniu elementów o mniejszych kluczach w górę drzewa i eliminacji jednego elementu z dołu struktury (liścia)
- Struktura stogu usprawnia sortowanie drzewiaste, ponieważ eliminuje niepotrzebne porównania elementu, który jest eliminowany z drzewa

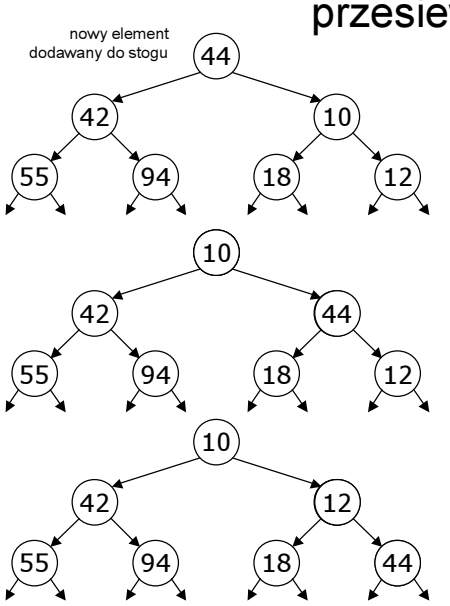
15

Stóg – reprezentacja tablicowa



16

Dodawanie elementów do stogu - przesiewanie

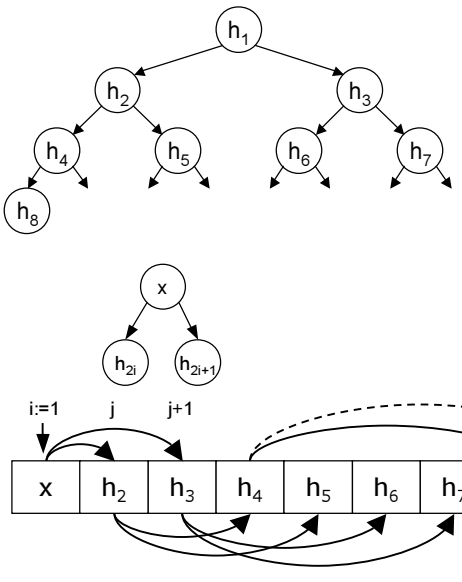


Dodanie elementu musi utrzymać warunek stogu (liście potomne są nie większe niż ich rodzic).

Nowy element wstawiany jest na wierzchołek drzewa, a następnie „przesiewany” przez węzły **mniejszych** elementów stogu, które podnoszą się przez to do góry.

17

Przesiewanie c.d.




```
PROCEDURE Przesiewanie(l,p : integer);
VAR
    i,j:integer;
    x: obiekt;
BEGIN
    i:=l; j:=2*i;
    x:=A[i];
    WHILE j<=p DO
    BEGIN
        IF j<p THEN
            IF A[j].klucz > A[j+1].klucz THEN
                j:=j+1;
        IF x.klucz > A[j].klucz THEN
        BEGIN
            A[i]:=A[j];
            i:=j; j:=2*i
        END
        ELSE j:=p+1; {stop}
    END;
    A[i]:=x;
END;
```

18

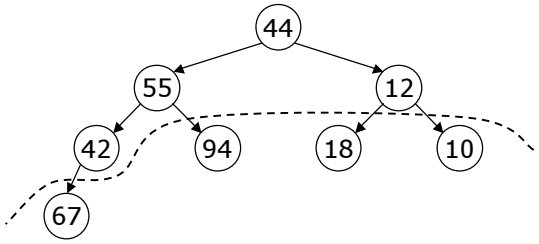
Tworzenie stogu z dowolnej tablicy

44	55	12	42	94	18	10	67

Połowa tablicy:

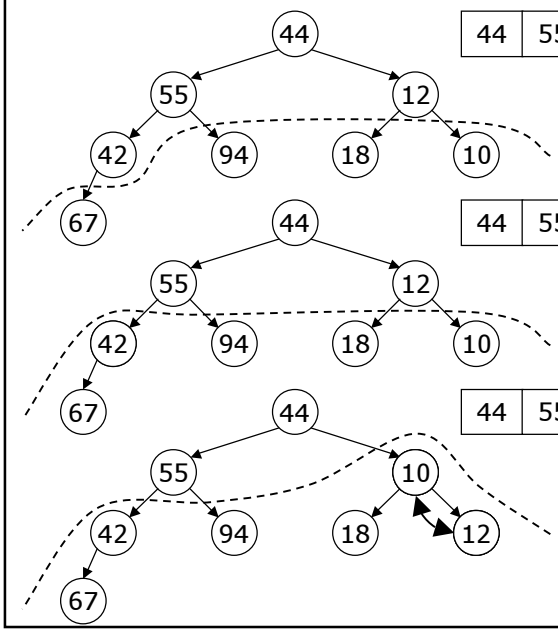
elementy *(n div 2)+1 ... N*

od początku tworzą stóg, gdyż nie zachodzi między nimi żadna relacja.



19

Tworzenie stogu c.d.



44	55	12	42	94	18	10	67

Kolejne elementy dodaje się do stogu metodą przesiewania

44	55	12	42	94	18	10	67

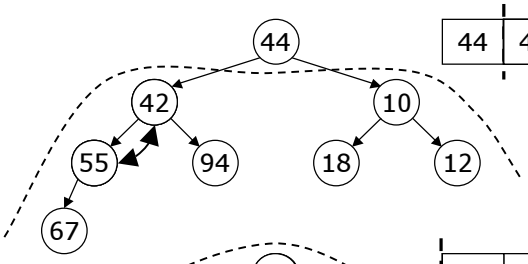
```
...
Przesiewanie(4,N);
...
```

44	55	10	42	94	18	12	67

```
...
Przesiewanie(3,N);
...
```

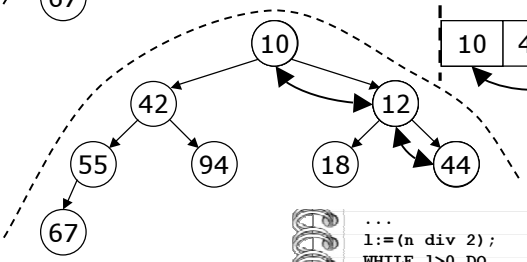
20

Tworzenie stogu c.d.



44	42	10	55	94	18	12	67

```
...
Przesiewanie(2,N);
...
```



10	42	12	55	94	18	44	67

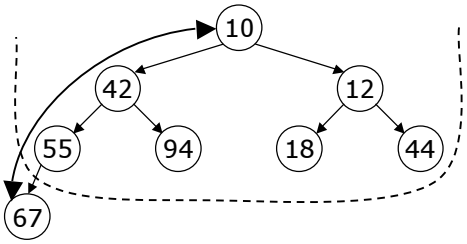
```
...
Przesiewanie(1,N);
...
```

```
...
l:=(n div 2);
WHILE l>0 DO
BEGIN
    Przesiewanie(l,n);
    l:=l-1;
END
...
```

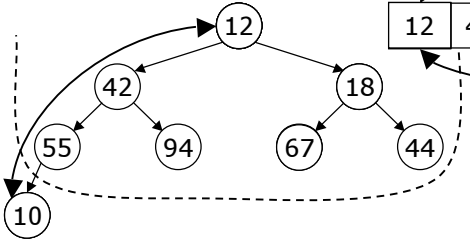
21

Sortowanie stogowe

Mając tablice a strukturze stogu, sortowanie polega na:



- pobraniu z wierzchołka elementu i usunięciu go ze stogu
- przesianiu przez zmniejszony stóg elementu ostatniego
- w miejsce ostatniego elementu umieszczenie elementu zdjętego z wierzchołka

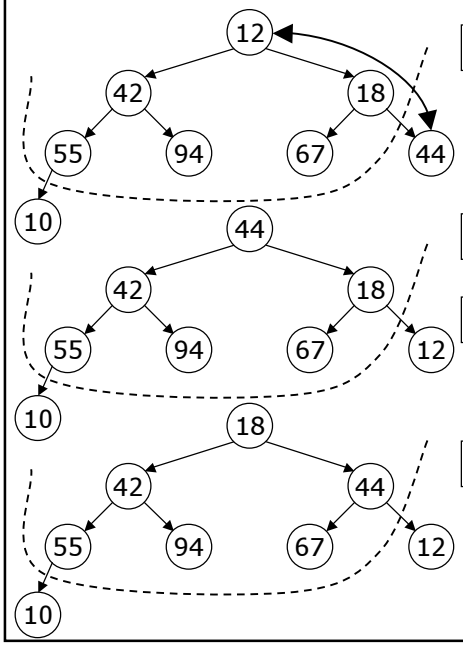


12	42	18	55	94	67	44	10

```
x:=A[1];
A[1]:=A[8];
A[8]:=x;
Przesiewanie(1,7);
```

22

Sortowanie stogowe c.d.



12	42	18	55	94	67	44	10

44	42	18	55	94	67	12	10

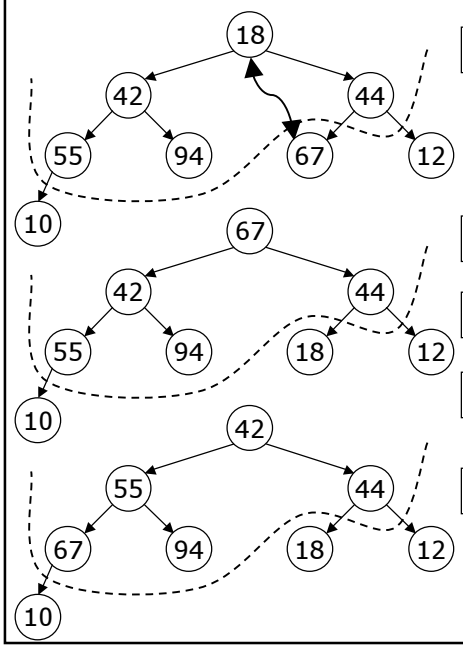
18	42	44	55	94	67	12	10

18	42	44	55	94	67	12	10

```
x:=A[1];
A[1]:=A[7];
A[7]:=x;
Przesiewanie(1,6);
```

23

Sortowanie stogowe c.d.



18	42	44	55	94	67	12	10

67	42	44	55	94	18	12	10

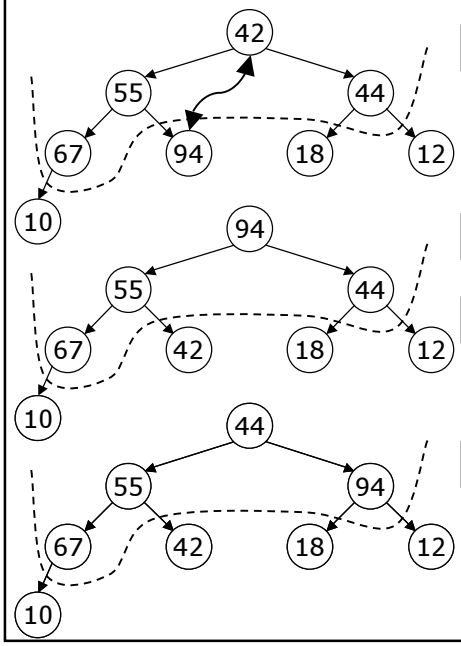
42	67	44	55	94	18	12	10

42	55	44	67	94	18	12	10

42	55	44	67	94	18	12	10

```
x:=A[1];
A[1]:=A[6];
A[6]:=x;
Przesiewanie(1,5);
```

Sortowanie stogowe c.d.



42	55	44	67	94	18	12	10

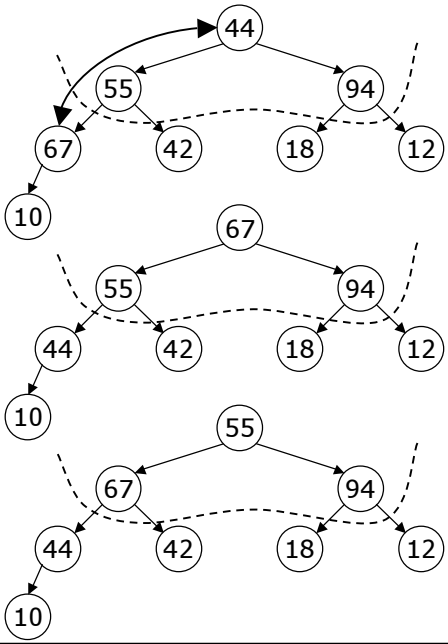
94	55	44	67	42	18	12	10

44	55	94	67	42	18	12	10

```
x:=A[1];
A[1]:=A[5];
A[5]:=x;
Przesiewanie(1,4);
```

25

Sortowanie stogowe c.d.



44	55	94	67	42	18	12	10

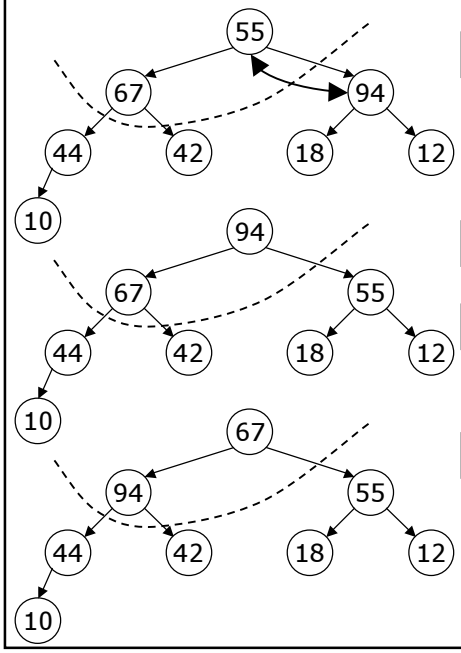
67	55	94	44	42	18	12	10

55	67	94	44	42	18	12	10

```
x:=A[1];
A[1]:=A[4];
A[4]:=x;
Przesiewanie(1,3);
```

26

Sortowanie stogowe c.d.



55	67	94	44	42	18	12	10

94	67	55	44	42	18	12	10

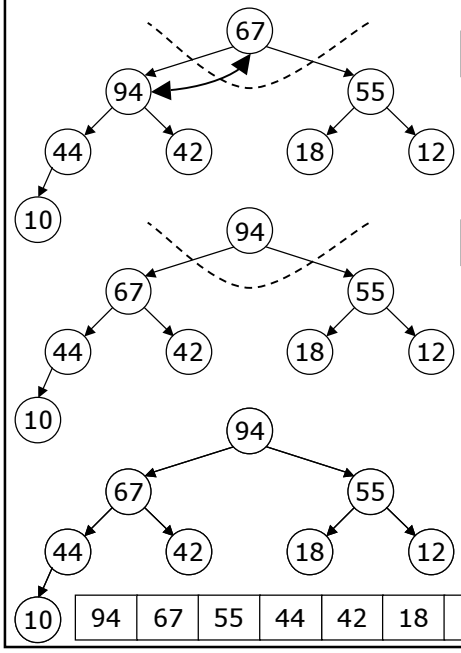
67	94	55	44	42	18	12	10

67	94	55	44	42	18	12	10

```
x:=A[1];
A[1]:=A[3];
A[3]:=x;
Przesiewanie(1,2);
```

27

Sortowanie stogowe c.d.



67	94	55	44	42	18	12	10

94	67	55	44	42	18	12	10

Po *n-1* krokach tablica będzie posortowana. Ponieważ z przesiewanie wymaga *log(n)* operacji, całe sortowanie stogu wymaga *n*log(n)* operacji.

94	67	55	44	42	18	12	10

```
p:=N;
WHILE p>1 DO
BEGIN
    x:=A[1];
    A[1]:=A[p];
    A[p]:=x;
    p:=p-1;
    Przesiewanie(1,p);
END
```

28

Algorytm sortowania stogowego - program

```
PROCEDURE Przesiewanie(l,p : integer);
VAR
   i,j:integer;
   x: obiekt;
BEGIN
   i:=l; j:=2*i;
   x:=A[i];
   WHILE j<=p DO
   BEGIN
      IF j<p THEN
         IF A[j].klucz > A[j+1].klucz THEN
            j:=j+1;
      IF x.klucz > A[j].klucz THEN
      BEGIN
         A[i]:=A[j];
         i:=j; j:=2*i
      END
      ELSE j:=p+1; {stop}
   END;
   A[i]:=x;
END;
```

```
PROCEDURE Sort_Stogowe;
VAR
   l,p:integer;
   x:obiekt;
BEGIN
   l:=(n div 2);
   WHILE l>0 DO
   BEGIN
      Przesiewanie(l,n);
      l:=l-1;
   END;
   p:=N;
   WHILE p>1 DO
   BEGIN
      x:=A[1];
      A[1]:=A[p];
      A[p]:=x;
      p:=p-1;
      Przesiewanie(1,p);
   END
END;
```

29

Sortowanie stogowe - wnioski

- Średnia efektywność metody: n*log(n)
- Efektywność metody dla najgorszego przypadku jest zbliżona do średniej !
- Duża liczba operacji przygotowawczych – nie zalecana dla małej liczby obiektów.

30