

- ▲ Proč se vůbec řazením zabýváme? Proč chceme posloupnosti dat řadit?
  
- ▲ Zkuste podrobně popsat vlastními slovy postup, jak seřadit:
  - 10 hracích karet do ruky při rozdávání
  - nastoupenou fotbalovou jedenáctku v dresech, pokud má předstoupit o krok dopředu seřazená podle čísel na dresech
  - 200 písemek podle abecedy, jsou-li k dispozici 4 pomocníci
  - lidi stojící ve frontě v úzké chodbě podle data narození (pokud každý může mluvit jenom se svým sousedem)
  
- ▲ Všechny asociativní řadící algoritmy patří do třídy  $\Omega(n \log_2 n)$ . Dokážete najít algoritmus, který jisté typy posloupností uspořádá rychleji, tj. v čase  $o(n \log_2 n)$ ?

▲ Na čem je založen princip fungování algoritmu Bubblesort? Jaká je jeho složitost v nejlepších, průměrných a nejhorších případech? Je tento algoritmus stabilní? Chová se přirozeně? Je 'in situ' algoritmus?

▲ Na čem je založen princip fungování algoritmu Insertsort? Jaká je jeho složitost v nejlepších, průměrných a nejhorších případech? Je tento algoritmus stabilní? Chová se přirozeně? Je 'in situ' algoritmus?

▲ Na čem je založen princip fungování algoritmu Selectsort? Jaká je jeho složitost v nejlepších, průměrných a nejhorších případech? Je tento algoritmus stabilní? Chová se přirozeně? Je 'in situ' algoritmus?

▲ Pomocí algoritmu Mergesort seřadte posloupnost [6,3,8,3,7,3,6,1,4].

▲ Pomocí algoritmu Quicksort seřadte posloupnost [6,3,8,3,7,3,6,1,4].

▲ Dojde v chování následujícího algoritmu (Dijkstrův quicksort) k zásadní změně pokud nahradíme test na řádce č. 8 testem  $i < j$ ?

```
1: procedure quicksort(l,r:integer)
2:   var i,j,a:integer;
3:   begin
4:     i:=l; j:=r; a:=prvek(l,r);
5:     repeat
6:       while A[i]<a do i:=i+1;
7:       while A[j]>a do j:=j-1;
8:       if i<=j then
9:         swap(A[i],A[j]);
10:        i:=i+1;j:=j-1;
11:     until i>j;
12:     if l<j then quicksort(l,j);
13:     if i<r then quicksort(i,r);
14: end;
```

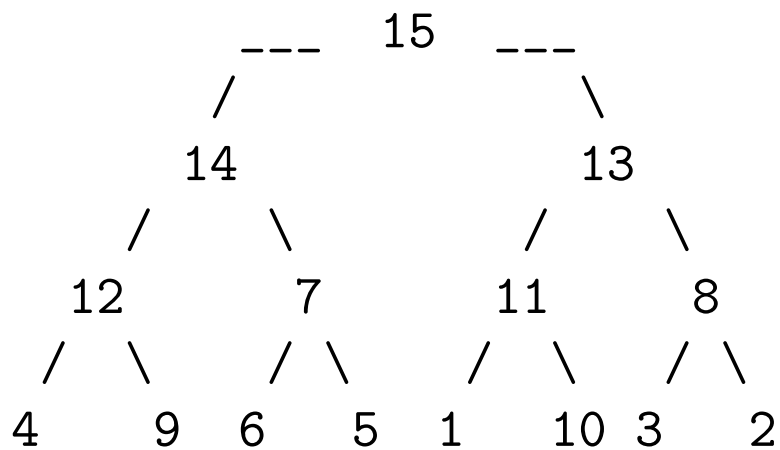
▲ Navrhněte nějaké způsoby výběru prvku a (řádek č. 4 v předchozím programu). Jakou vlastnost má nejvhodnější prvek?

▲ Def: Necht'  $K$  je úplně uspořádaná množina tzv. klíčů (např. z  $N$ ).

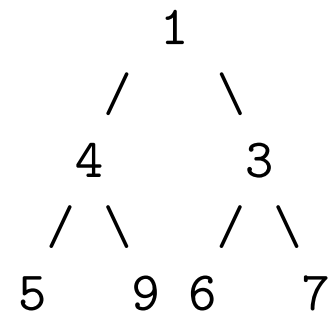
*Binární halda* je binární strom, jehož uzly jsou ohodnoceny prvky  $K$  a který splňuje:

1. Délky všech větví se liší nejvýše o 1: mají délku  $k$  nebo  $k - 1$  ( $k$  - hloubka stromu)
2. Hodnoty uzlů na každé větvi jsou vzestupně (sestupně) uspořádány.

Příklady

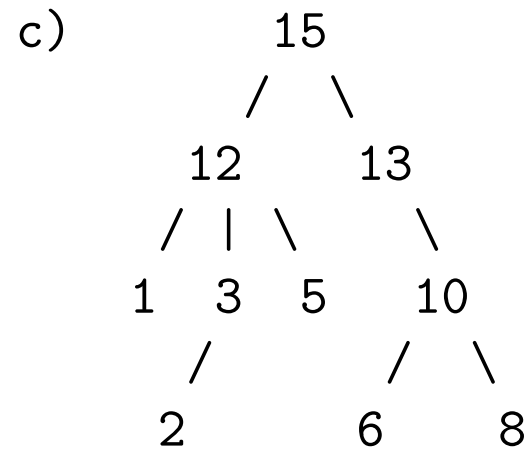
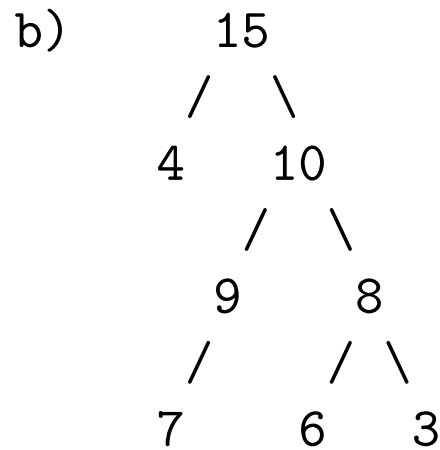
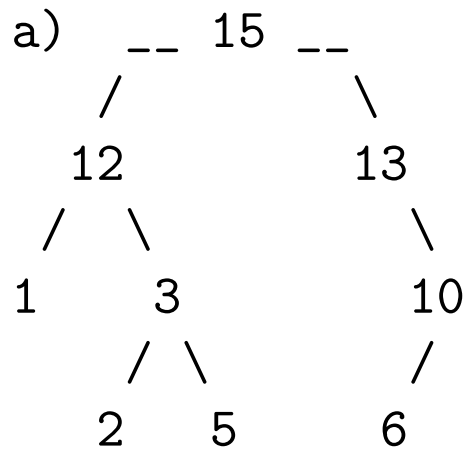


maximová halda

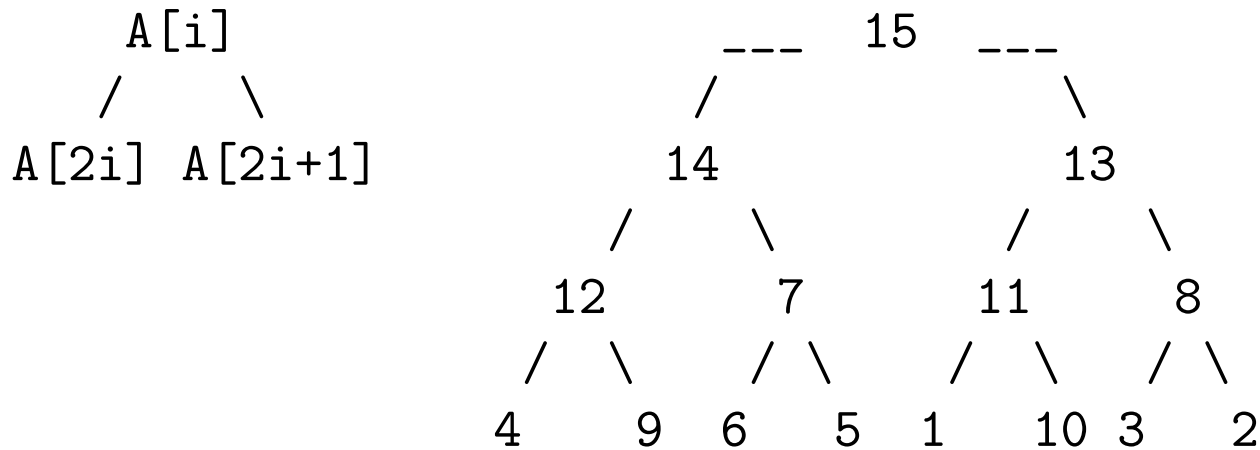


minimová halda

▲ Rozhodněte, zda jsou následující stromy binární haldou. Odpověď zdůvodněte.



▲ Přepište tuto haldu do pole  $A[1..n]$ ,  $n=15$ , podle předpisu:  
pro každé  $1 \leq i \leq n$



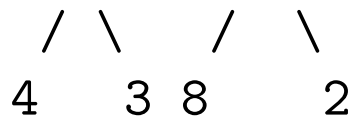
▲ Ze dvou maximových hald velikosti  $n$  vytvořte jednu velikosti  $2n + 1$  přidáním určeného prvku:

a) 7            3    přidejte 9

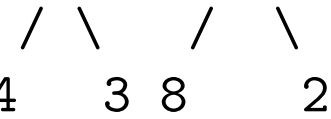
b) 7            3    přidejte 5

c) 7            3    přidejte 1

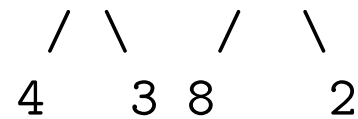
d) 5            10   přidejte 11



e) 5            10   přidejte 9

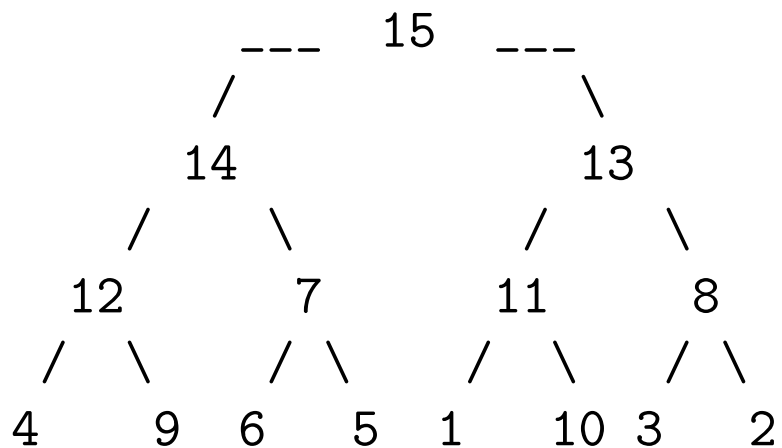


f) 5            10   přidejte 6



▲ Vytvořte binární maximovou (zleva zarovnanou) haldu z následujícího pole [5,9,1,14,7,11,13,4,12,6]

▲ Odeberte z následující (maximové) haldy maximum a vytvořte opět (zleva zarovnanou) haldu:



▲ Pomocí algoritmu Heapsort seřadte posloupnost [6,3,8,3,7,3,6,1,4]. Použijte in situ variantu algoritmu.

▲ Zkuste neformálně postihnout podstatu následujícího algoritmu.

```
procedure s1(A[0..N-1]);
begin
  left = 0; right = N - 1; swapped = true;
  while (swapped == true)
    swapped = false;
    for (i = left; i < right; i = i + 1)
      if (A[i] > A[i + 1]) then
        swap(A[i], A[i + 1]);
        swapped = true;
    right = right - 1;
    for (i = right; i > left; i = i - 1)
      if (A[i] < A[i - 1]) then
        swap(A[i], A[i - 1]);
        swapped = true;
    left = left + 1;
end;
```



▲ Který algoritmus řazení je nejlepší?

Název	Časová složitost		
	Min	Prům	Max
bubblesort - řaz. výměnou	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$
heapsort - řaz. haldou	$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$
insertsort - řaz. vkládáním	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$
mergesort - řaz. slučováním	$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$
quicksort - řaz. rozdělováním	$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$	$\Theta(n^2)$
selectsort - řaz. výběrem	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$

▲ Pokud známe uspořádání dvojic  $(a, b)$  a  $(b, c)$ , můžeme něco usoudit i o dvojici  $(a, c)$ ? Ve kterých případech? Uveďte příklady využití v algoritmech řazení.

▲ Jak vypadají nejvíce (nejméně) příznivá vstupní data pro algoritmus `insertsort`?

```
procedure insertsort(A[0..n-1]);
int i, j, temp;
begin
  for i: = 1 to n-1 do
    temp := A[i];
    j := i;
    while ((j > 0) and (A[j-1] > temp)) do
      A[j] := A[j-1];
      j := j-1;
    A[j] := temp;
end;
```

▲ Porovnejte následující programy xsort a ysort. Jaký koncept mají společný? V čem se zásadně liší?

```
procedure xsort A[low,high]:  
  if low<high then  
    middle:=round((low+high)/2);  
    xsort A[low,middle];  
    xsort A[middle+1,high];  
    seřaď seřazené posloupnosti A[low,middle] a A [middle+1,high];
```

```
procedure ysort A[low,high]:  
  if low<high then  
    zvol(pivot);  
    rozděl A na část prvků menších než pivot A[low,ip-1]  
    a část větších než pivot A[ip+1,high];  
    ysort A[low,ip-1];  
    ysort A[ip+1,high];  
    spoj do jedné posloupnosti A[low,ip-1], pivot, A[ip+1,high];
```

## Algoritmy mergeSort a quickSort – funkcionální varianta:

```
mergeSort    :: [Int]->[Int]
mergeSort [] = []
mergeSort [x] = [x]
mergeSort s = merge (mergeSort u) (mergeSort v)
              where (u,v) = splitAt (n `div` 2) s
                    n = length s

merge s [] = s
merge [] t = t
merge (x:u)(y:v) = if x <= y then x:merge u (y:v)
                  else y:merge (x:u) v

quickSort    :: [Int]->[Int]
quickSort [] = []
quickSort (p:t) = quickSort lt ++ [p] ++ quickSort gs
                where lt = [x | x<-t, x<p]
                      gs = [x | x<-t, x>=p]
```

## Algoritmy mergeSort a quickSort – imperativní varianta:

```
var A[1..n]:integer;
```

```
procedure mergeSort(p,r:integer);  
var q:integer;  
begin  
  if p<r then  
    q:=round((p+r)/2);  
    mergeSort(p,q);  
    mergeSort(q+1,p);  
    merge(p,q,r)  
end;
```

```
procedure merge(p,q,r:integer);  
var i,j,k:integer;  
var B[1..n]:integer;  
begin  
  i:=p; j:=q+1;  
  for k:=p to r do  
    if i<=q and (j>r or A[i]<=A[j])  
    then  
      B[k]:=A[i];  
      i:=i+1;
```

```
procedure quickSort(p,r:integer);  
var q:integer;  
begin  
  if p<r then  
    q:=part(p,r);  
    quickSort(p,q-1);  
    quickSort(q+1,r)  
end;
```

```
function part(p,r:integer):integer;  
var i,j,x:integer;  
begin  
  x:=A[r];  
  i:=p-1;  
  for j:=1 to r-1 do  
    if A[j]>=x then  
      i:=i+1;  
      swap(A[i],A[j]);  
  swap(A[i+1],A[r]);
```

```
else                                     return(i+1);
    B[k]:=A[j];                          end;
    j:=j+1;
    for k:=p to r do A[k]:=B[k];
end;
```