Corso di Algoritmi e Strutture Dati
Anno Accademico 2017–2018
Prof. Giovanni Pighizzini

Pseudocodice

F. Heapsort

F.1. Risistema (fixHeap)

```
Procedura risistema (heap H)
  v ← H
  x ← v.dato            // chiave da ''far scendere'' nella posizione appropriata
  daCollocare ← true
  do
    if v è una foglia then
      daCollocare ← false     // la posizione appropriata per x è stata trovata
    else
      u ← figlio di v di valore massimo
      if u.dato > x then
        v.dato ← u.dato                                  // la chiave in u risale
        v ← u                                                 // si prosegue su u
      else
        daCollocare ← false  // la posizione appropriata per x è stata trovata
  while daCollocare
  v.dato ← x                                        // copia x nella posizione trovata
```

F.2. Creazione dello heap (versione ricorsiva)

```
Funzione creaHeap (albero T)
  /* Trasforma l'albero binario quasi completo T in uno heap                  */
  if T ≠albero vuoto then
    creaHeap(T.sx)
    creaHeap(T.dx)
    risistema(T)
```

F.3. Creazione dello heap (versione iterativa)

```
Funzione creaHeap (albero T)
  /* Trasforma l'albero binario quasi completo T in uno heap */
  h ← altezza di T
  for p ← h downto 0 do
      foreach nodo n di profondità p do
          risistema(sottoalbero di radice n)
```

F.4. Schema di Heapsort

```
Algoritmo heapSort (array A)→ lista
  crea uno heap H da A
  X ← lista vuota
  while H ≠ ∅ do
      aggiungi all'inizio di X il valore presente nella radice di H
      colloca nella radice di H il valore che si trova foglia più in basso a destra
      rimuovi tale foglia
      risistema(H)
  return X
```

F.5. Risistema (fixHeap) su array posizionale (confrontare con F.1)

```
Procedura risistema (array A, intero r, intero l)
  /* A è un array i cui primi l elementi formano uno heap. Risistema la parte
     di heap la cui radice si trova alla posizione di indice r di A.
     Note: Il primo indice dell'array A è 0. Pertanto i figli di un nodo di
     indice i, se presenti, si trovano alle posizioni di indice 2i+1 e 2i+2.
     Quando viene richiamata all'interno di heapsort, il parametro l (numero di
     elementi nello heap, cioè ancora da ordinare) coincide con l'indice di A
     in cui inizia la porzione che è già stata ordinata.                  */
  v ← r
  x ← A[v]              // chiave da ''far scendere'' nella posizione appropriata
  daCollocare ← true
  do
      if 2 * v + 1 ≥ l then                              // v è l'indice di una foglia
          daCollocare ← false     // la posizione appropriata per x è stata trovata
      else
          u ← 2 * v + 1                                        // indice figlio sinistro
          if u + 1 < l and A[u + 1] > A[u] then     // c'è figlio destro ed è maggiore
              u ← u + 1                                                 // del sinistro
          // ora u contiene l'indice del figlio di v di valore massimo
          if A[u] > x then
              A[v] ← A[u]                              // la chiave in posizione u risale
              v ← u                                 // si prosegue sul figlio selezionato
          else
              daCollocare ← false  // la posizione appropriata per x è stata trovata
  while daCollocare
  A[v] ← x                                             // copia x nella posizione trovata
```

F.6. Creazione di uno heap rappresentato implicitamente in un array (confrontare con F.3)

```
Funzione creaHeap (array A)
  n ← lunghezza di A
  for i ← ⌊n/2⌋ downto 0 do
      risistema(A, i, n)
```

F.7. Heapsort (indici dell'array da 0)

```
Algoritmo heapSort (array A)→ lista
  creaHeap(A)
  for l ← lunghezza di A − 1 downto 1 do
      scambia A[0] e A[l]
      risistema(A, 0, l)
```