Mergesort e Heapsort

2º Trabalho PLC

Sumário

- Mergesort
- Heapsort

Mergesort

- Dividir-para-conquistar
- Complexidade de tempo: $\Theta(n \log_2 n)$
- Algoritmo criado por Von Neumann em 1945.
- Gasto extra de memória

Mergesort – Implementação

```
split :: [Int]->([Int],[Int])
split [] =([],[])
split [x]=([x],[])
split (x:y:xs) =(x:fst (split xs),y: snd (split xs))

merge :: [Int]->[Int]->[Int]
merge [] ys = ys
merge xs [] = xs
merge (x:xs) (y:ys)
         | x<y = [x]++(merge xs ([y]++ys))
     | otherwise = [y]++(merge ([x]++xs) ys)

mergesort:: [Int]-> [Int]
mergesort [] = []
mergesort [x] = [x]
mergesort xs = merge (mergesort (fst (split xs))) (mergesort (snd (split xs)))
```

Heapsort

- Utiliza uma estrutura de dados chamada heap.
- Complexidade de tempo: $\Theta(n \log n)$
- Foi desenvolvido em 1964 por **Robert W. Floyd** e **J.W.J. Williams**.
- Não é um algoritmo de ordenação estável.

Heapsort – Implementação

```
add:: Int->[Int]->[Int]
add n hp| hp==[] = [n]
                | otherwise = (bubbleUp (hp++[n]) ((length hp)) 0)


bubbleUp :: [Int]->Int->Int->[Int]
bubbleUp [] l p  = []
bubbleUp (x:xs) l p | xs==[] || ((l-p)-1) >= (length  xs)= (x:xs)
                                        |((2*p)+1) == (l) && (x < (xs!!((l-p)-1))) = ([(xs!!((l-p)-1))]++(take (l-p-1) xs))++[x]++(drop (l-p) xs)
                                        |((2*p)+2) == (l) && x < (xs!!((l-p)-1)) = ([xs!!((l-p)-1)]++(take (l-p-1) xs))++[x]++(drop (l-p) xs)
                                        |((2*p)+1) == (l) || ((2*p)+2) == (l) = x:xs
                                        | (l)>((2*p)+2) = (bubbleUp (bubbleUp (x:(bubbleUp xs l (p+1))) ((2*p)+1) p) ((2*p)+2) p)
                                        | (l)>((2*p)+1) = (bubbleUp (x:(bubbleUp xs l (p+1))) ((2*p)+1) p)
                                        | otherwise =  x:(bubbleUp xs l (p+1))


remove::[Int]-> [Int]
remove [] = []
remove [x] = []
remove (x:y:z:xs) | z>y = (add y (z:xs))
                                  | otherwise = (add z (y:xs))
remove [x,y] = [y]


sort ::[Int]->[Int]
sort hp | hp == [] = []
                | otherwise = (sort (remove hp))++[(head hp)]


makeheap:: [Int]->[Int]->[Int]
makeheap [] hp  = hp
makeheap (x:xs) hp = makeheap xs (add x hp)

heapsort:: [Int]->[Int]
heapsort l1 = sort (makeheap l1 [])
```