Tuesday, September 08, 2009
3:33 PM

Quicksort (worst case)

$T(n) = T(n-1) + cn$ — list split "unevenly"

$k=0 \quad = cn + T(n-1)$

$\vdots$

$k=3 \quad = 4cn - c\sum_{k=0}^{n} k + T(n-4)$

$= c(k+1)n - c\sum_{k=0}^{n} k + T(n-k+1)$

want this $T(1)$

$\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$

Let $k=n$

$= c(n^2+n) - \frac{cn^2}{2} - \frac{cn}{2} + T(1)$

$= \frac{cn^2}{2} - \frac{cn}{2} \leftarrow O(n^2)$ → happens when pivot is smallest element (degenerate case)

Best case

$T(n) = 2T\left(\frac{n}{2}\right) + cn$

$k=1 \quad = cn + 2T\left(\frac{n}{2}\right)$

$\vdots$

$= k(cn) + 2^k T\left(\frac{n}{2^k}\right)$

let $2^k = n \Rightarrow k = \lg n$

$T(n) = cn\lg n + nT(1) \in O(n\lg n)$

Tuesday, September 08, 2009
3:41 PM

More on sorting ...

STL implementation of insertion sort & ASG2: heapSort

↓

standard Template Library vector<> (& list<>)

└ both containers

```
template <typename T>
void insertionSort (vector<T>& a)
{
  int j;
  for (int p = 1; p < a.size(); p++) {
    T tmp = a[p];
    for (j = p; j > 0 && tmp < a[j-1]; j--)
      a[j] = a[j-1];
    a[j] = tmp;
  }
```

fcn works for vector<T>; will it work Array<T>? yes, need

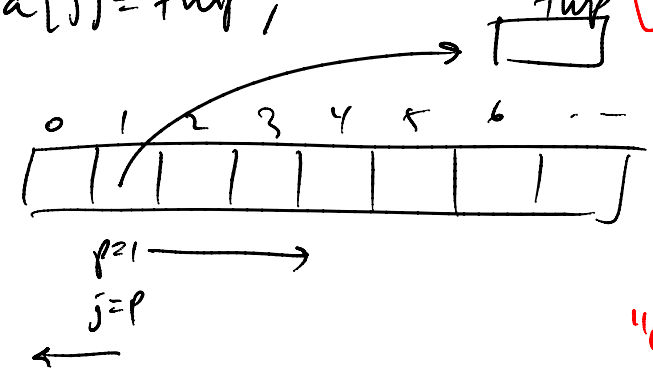
(vector is like your Array<>)

need to provide operator[]

O(n) O(n-1)

pairwise comparison sort

need to have operator< defined for type T (e.g. if T were "Employee" Employee::operator<())



O(n(n-1)) ∈ O(n²)

1 "in"

iteration

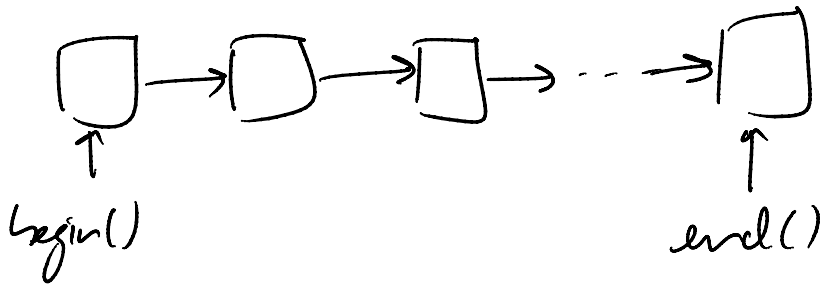
1
2
3

a[1] gets inserted in a[0] if a[1] < a[0]
a[2] " " " a[0..1] if a[2] < ...
a[3] " " " a[0...2] if ...

Tuesday, September 08, 2009
3:57 PM

- what if operator[] doesn't exist for given class?
  e.g. linked list

- for such classes, to iterate through their elements
  use iterators — like pointers



- analogous sort routine:

```
template <typename iterator, typename T>
void insertionSort(const iterator& begin,
                   const iterator& end,
                   const T& obj)
{
    for (iterator p = begin+1; p != end; p++)
    {
        T tmp = *p;
        for (iterator j = p; j != begin && tmp < *(j-1); --j)
            *j = *(j-1);
        *j = tmp;
    }
```

(each node holds)

typename T must provide operator=

typename T must provide operator<

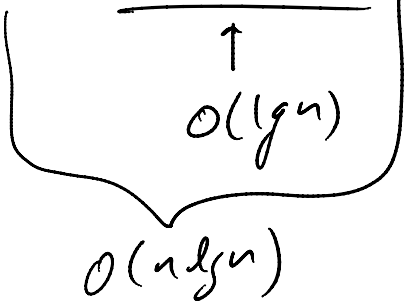
- we'll revisit these later, for now, know that you can access an STL vector two ways

1. for (int i=0; i< a.size(); i++)
   a[i] = ...

2. for (std::vector<int>::iterator ap = a.begin();
   ap != a.end(); ap++)
   (*ap) = ...
   (
   & do this because by *ap type might have [] operator: (*ap)[]
   (can subscript then *ap[])

Tuesday, September 08, 2009
4:09 PM

- insertion sort is ok but $O(n^2)$
- HeapSort has best big-oh run time ($O(n \lg n)$)
  1. construct a heap of n elements $O(n)$
  2. perform n deleteMin() operations



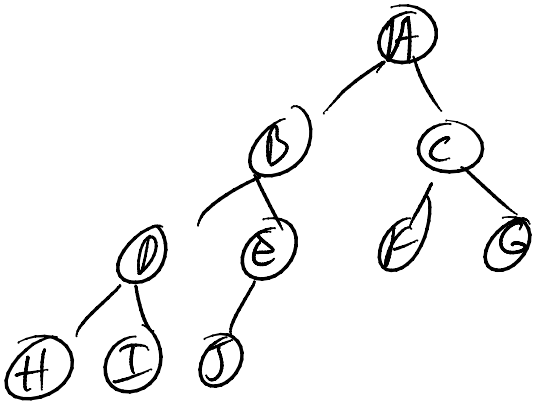
- §3.3 members of STL vector:

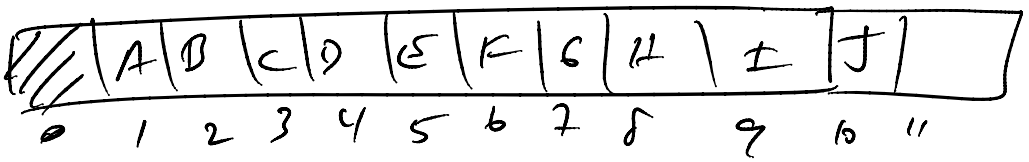
int	size()	status queries
void	clear()	remove all
bool	empty()	

- addition / deletion of elements:
  - push_back() : add element to back of list
  - pop_back() : remove "
- operators
  - operator[]
- iterators
  - vector<int>::iterator
  - begin() end()
- iterator methods ++, --

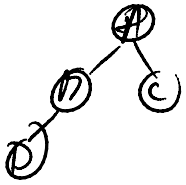
- your BinHeap class:
  - should provide some of the functions above (names may be slightly different)
    e.g. top() — findMin()
    pop() — deleteMin()
  - can use your own Array<int> or vector<int>
    (Array<int> / vector<int> already templated)

- implement binary heap (§6.3)
- binary heap: complete binary tree represented by an array



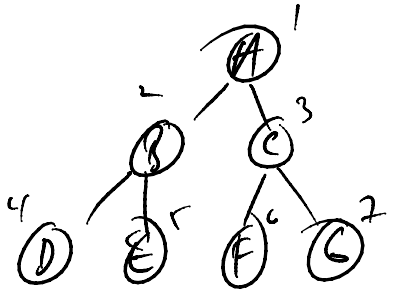


- with height of tree h, $2^h \le n \le 2^{h+1} - 1$ nodes



$\{ h = 2,\ 4 \le n \le 4$

- for n nodes, height of tree is $\lg n$
- for any element in array position i
  left child is at position $2i$
  right child is at " $2i+1$
  parent is at $\lfloor i/2 \rfloor$
  (floor, say, we have integer division $i/2$



pos 3, $l = 2(3) = 6$ (F)
$r = 2(3)+1 = 7$ (G)
$p = 3/2 = 1$ (A)

Tuesday, September 08, 2009
4:25 PM

- heap order property: smallest element always at top (root) (don't have to sorted)

  ⇒ findMin ∈ O(1) constant-time op. (arr[1])

- for any node, it's value (key) should be smaller than all of its children.

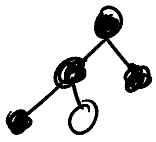
- insert() & deleteMin() require a bit of work.

- insert(x):

  1. create a 'hole' in the next available location

     int hole = ++n;

     ↑ keep count of # of elements



  2. if x can be placed in hole without violating order, do so & exit

     else

     move hole up to parent (slide parent down into hole)

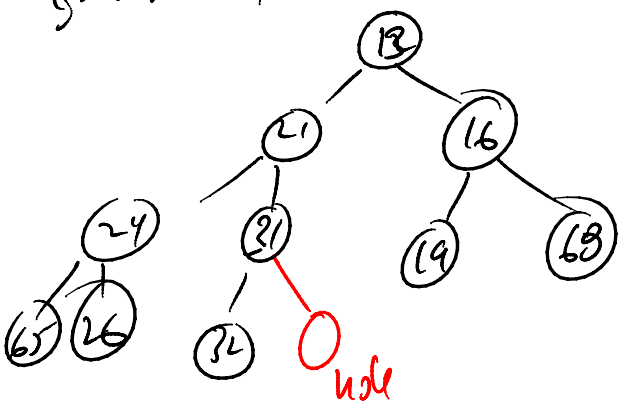
     arr[hole] = arr[hole/2];

     bubble up (percolate up) the hole towards root

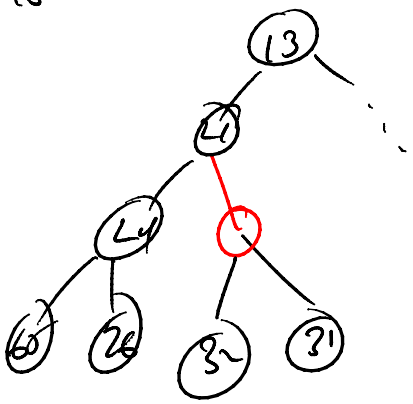
     continue until x can be placed into hole

e.g. insert 14 ; 14 < 31? yes, percolate up

e.g. insert 14



is 14 < 31? yes, percolate up

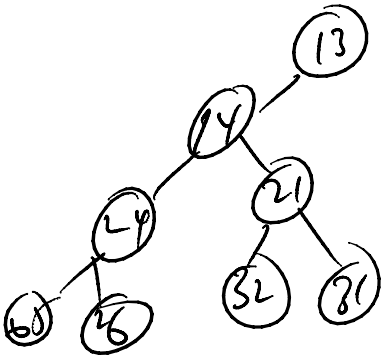


is 14 < 21? yes, percolate up

is 14 < 13? No, stop

hole moves up towards root

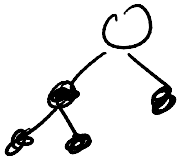
- stopping criteria is pos == 1



- Heap's insertion.
- deletion:

```
T deleteMin()
{
    if(empty()) return -1; // error
    T tmp = arr[1];
    arr[1] = arr[n--];
    percolateDown(1)
}
```

↑ location of hole



return (true);

}

Tuesday, September 08, 2009
4:38 PM

- other ops: key at position p; increment key by this much

incKey(p, Δ) : percolate down

decKey(p, Δ) : percolate up