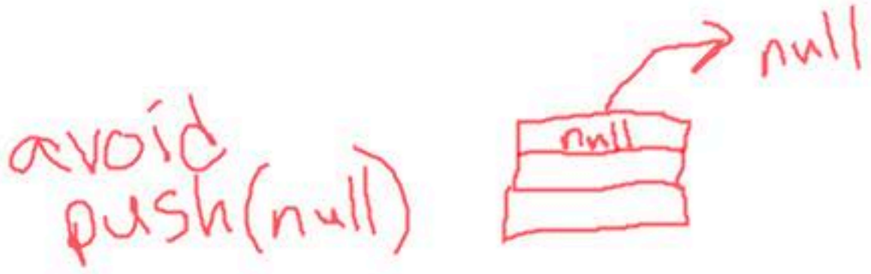
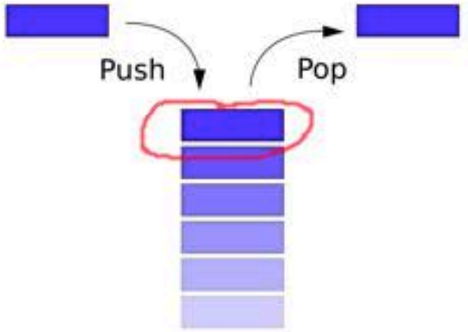




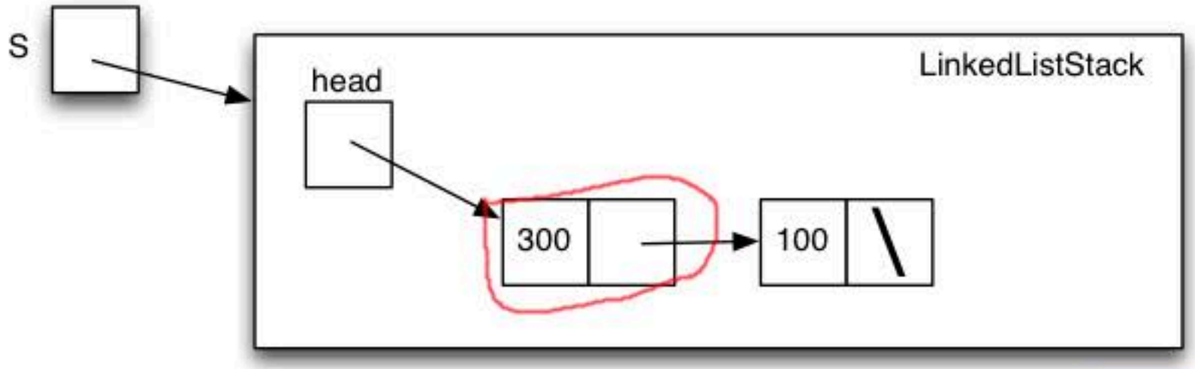
pop
null

```
public interface Stack {
    /* put the element on the top of the stack */
    public void push(Object ele);

    /* remove the element on top of the stack
    and return it; Returns null if stack is empty
    */
    public Object pop();

    /* return the element on top of the stack; returns null
    if stack is empty */
    public Object peek();
}
```

LinkedListStack

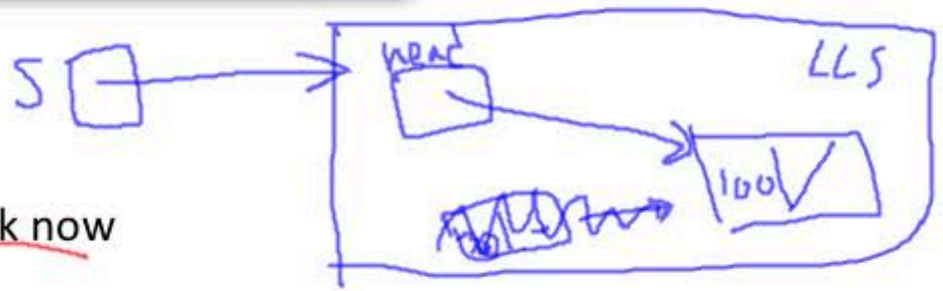
```
Stack s = new LinkedListStack();
```


```
s.push(100);
s.push(300);
```



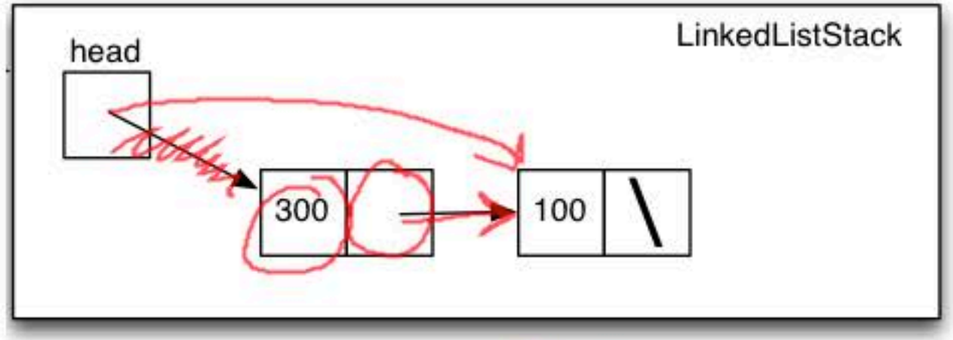

```
s.pop();
```

draw the LinkedListStack now



Empty case []

Non-empty case [300, 100]




```
/* remove the element on top of the stack
and return it; Returns null if stack is empty
*/
public Object pop();
```

```
if (head == null) return null;
Object r = head.data;
else head = head.next;
}
return r;
}
```

r 300

head

100

r 100

head

null

Code to run the experiment

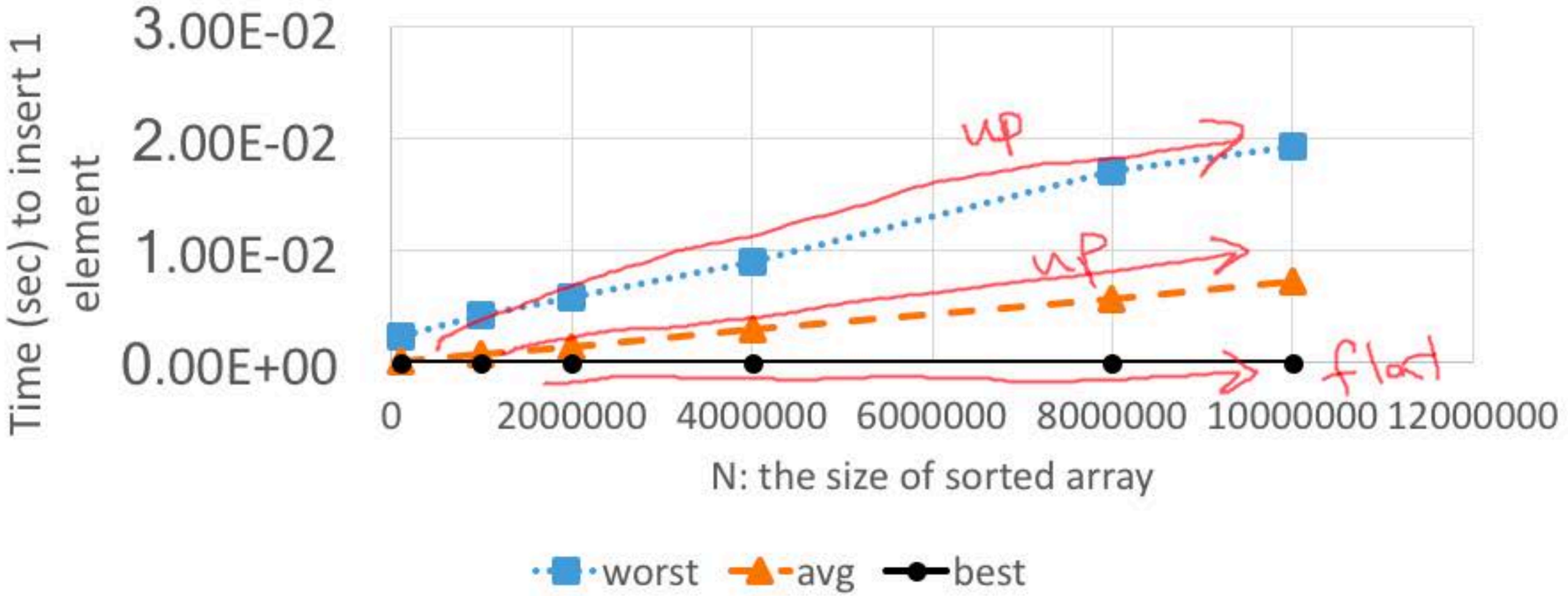
```
// create an sorted array of N random integers; N given by command line argument
System.out.println("creating and sorting array");
final int N = Integer.parseInt(args[0]);
final Random r = new Random();
final int[] array = r.ints(N).toArray();
Arrays.sort(array);

// generate 1000 random integers to insert into the sorted array
System.out.println("creating test inputs");
final int T = 1000;
final int[] trialElements = r.ints(T).toArray();
final long[] results = new long[T];

// run 1000 experiments
System.out.println("running experiments");
for (int t=0; t<T; t++) {
    long trial_start = System.nanoTime();
    // always just insert at the end
    insertSorted(array, N-1, trialElements[t]);
    long trial_end = System.nanoTime();
    results[t] = trial_end - trial_start;
}
```

not timed

not timed

each iteration has a time

Can we find an analytical model for worst/avg/best case time as a function of N?



$$worstTime(N) = c * N$$ yes

$$avgTime(N) = \frac{c}{2} * (N + 1)$$ yes

$$bestTime(N) = c * 1$$ no

c = time it takes to to do 1 swap

hypothesis: insertion time higher for bigger N

Common functions in algorithm analysis

constant	logarithm	linear	n log n	quadratic	cubic	polynomial	exponential
1	$\log n$	n	$n \log n$	n^2	n^3	n^d	a^n

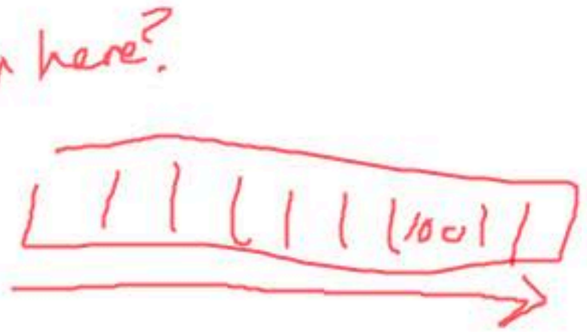
$a > 1$

slow growth

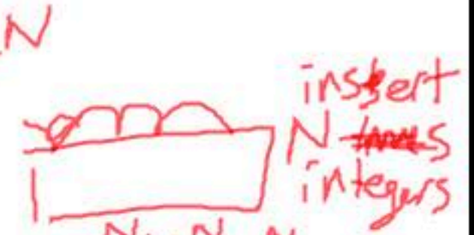
fast growth

from Goodrich, Tamassia, Goldwasser

Examples



- Problems with running time **linear** in N
  - search an unsorted array of size N for a value
- Problems with running time **quadratic** in N
  - sort N students using our `insertSorted` algorithm (put new element at end of array and swap into sorted position)
- Problems with running time **exponential** in N
  - find the quickest route for a UPS truck with N packages to deliver to the destinations



Big-Oh

$R(n)$ = running time for input size N

$R(n) \in O(g(n))$ means g(n) grows same or faster than $R(n)$ up to constant factors

$\exists k > 0 . R(n) \leq k * g(n)$ for all $n >= n_0$

$R(n) = 2n^2 + 5n\log n + 50$

prove $R(n) \in O(n^2)$

$50 + 5 + 2 = 57$

pick $k = 57$

pick $n_0 = 1$

$2n^2 + 5n\log n + 50 \leq 57n^2$

$n = 1$ $2 + 0 + 50 \leq 57$

$52 \leq 57$

what is n_0? Just a sufficiently large number