```
typedef struct CAList {
  int size, start, capacity;
  int* contents;
} CAList;


CAList* make_alist(int start_capacity) {
  CAList* alist = calloc(1, sizeof(CAList));
  alist->size = 0;
  alist->start = 0;
  alist->capacity = start_capacity;
  alist->contents = calloc(start_capacity, sizeof(int));
  return alist;
}

int indexFor(CAList* alist, int index) {
  int ans = (alist->start + index) % alist->capacity;
  printf("Index for %d is %d\n", index, ans);
  return ans;
}

void expandCapacity(CAList* alist) {









}

int get(CAList* alist, int index) {
  // ASSUME index is in bounds
  int toLookup = indexFor(alist, index);
  return alist->contents[toLookup];
}

void prepend(CAList* alist, int value) {
  if(alist->size >= alist->capacity) { expandCapacity(alist); }
  alist->size += 1;
  alist->start = alist->start - 1;
  if(alist->start == -1) { alist->start = alist->capacity - 1; }
  alist->contents[alist->start] = value;
}

void add(CAList* alist, int value) {
  if(alist->size >= alist->capacity) { expandCapacity(alist); }
  alist->contents[indexFor(alist, alist->size)] = value;
  alist->size += 1;
}


void print_alist(CAList* calist) {
  for(int i = 0; i < calist->capacity; i += 1) {
    printf("%d ", calist->contents[i]);
  }
  printf("\n");
}

int main(int argc, char** args) {
  CAList* a = make_alist(30);
  print_alist(a);
  prepend(a, 30);
  print_alist(a);
  add(a, 40);
  print_alist(a);
  prepend(a, 20);
  print_alist(a);
  add(a, 70);
  print_alist(a);
}
```

Index for 0 is 29

Amortized O(1) 1+2+4+8....
(Average over many calls)

1
1
1
1

(29 + 1) % 30
0

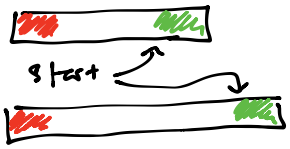
What index will 30 be stored at?

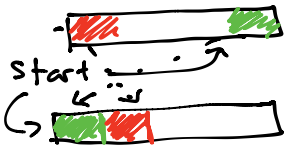
A: 0 B: 1

C: 30 D: 29

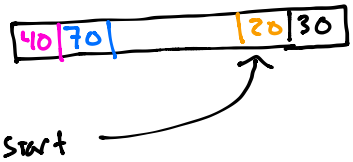
E: Something else

Discuss expandCapacity





size: 0 1 2 3 4
cap: 30



get(a, 0)
=> 30

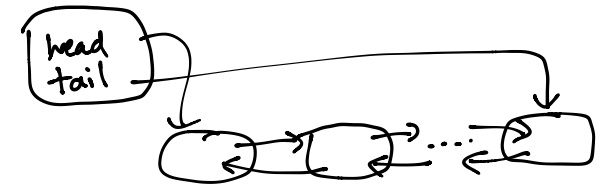
CircularArrayList

```
E get(int index) {
    return this.contents[index];
}
```

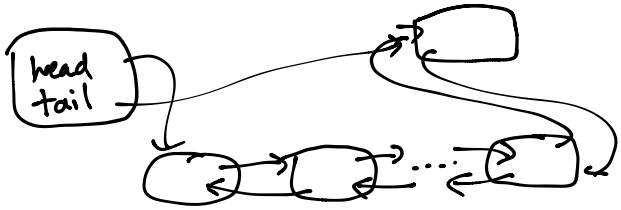
	get(index)	add(val) (add at end)	prepend(val)	remove(val)
AList	Worst: O(1) Best: O(1)	Worst: O(n) Best: O(1) Average: O(1)	Worst: O(2n) O(n) Best: O(n) Average: O(n)	
CAList (front of sheet)			Worst: O(n) Best: O(1) Ave: O(1)	
LList	Best: O(1) Worst: O(n) Average: O(n)	Best: O(n) Worst: O(n)	Best: O(1) Worst: O(1)	
DLList		Worst: O(1)		

```
typedef struct Node Node;
struct Node {
  Node* next;
  Node* prev;
  int val;
};

struct DLList {
   Node* head;
   Node* tail;
   int size;
}
```



add-at-end



Discussion Tuesday:

- Practice DS exam
- Can't take materials with you