# More dynamic memory

In addition to new, we also have delete

delete – let system know this memory is no longer in use.

– Almost always, a call to new should have a corresponding call to delete somewhere afterward.

Example: allocate array of variable size:

```
int* A = new int[n];
A[0] = ...
  :
  :
// done with A; let libc know:
delete [] A;
```

use these whenever you delete an array.

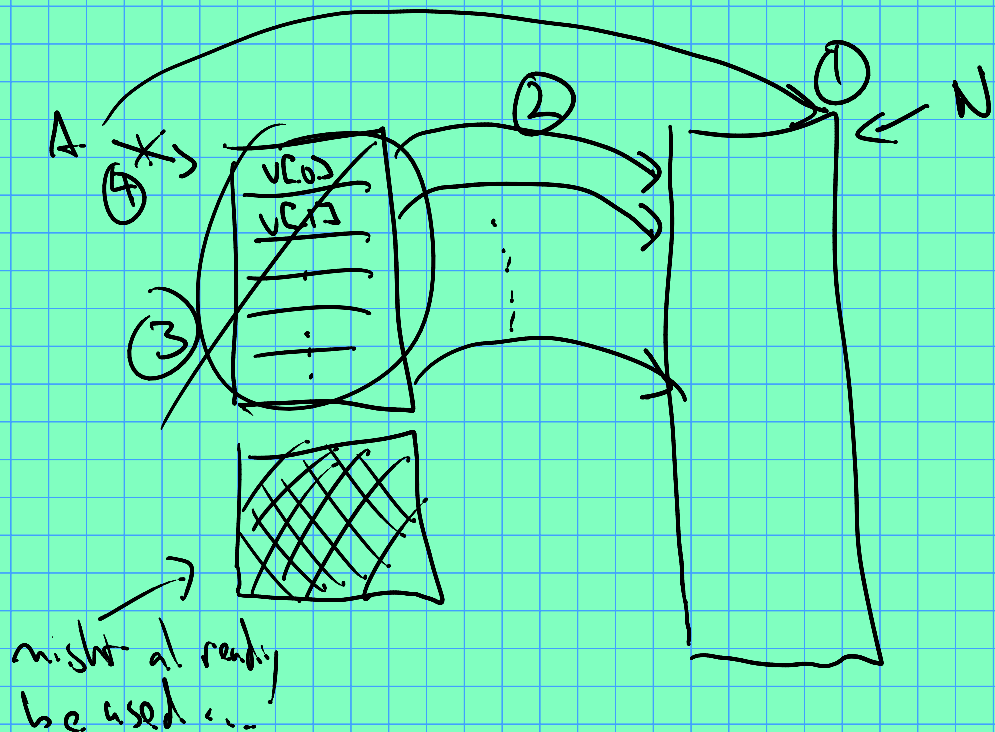
Exercise: how do vectors grow?

Seems to double the allocation (capacity) whenever it runs out of space.

Question: How many steps are required to perform n push_backs into an initially empty vector?

How about if the size only grew by 1 (instead of doubling)?

Before we can figure this out, how is the vector being grown?




To allocate more space:

1. Make a new array.
2. Copy elements from old to new
3. delete the old one!
4. redirect pointer A

Let's put this into a function:

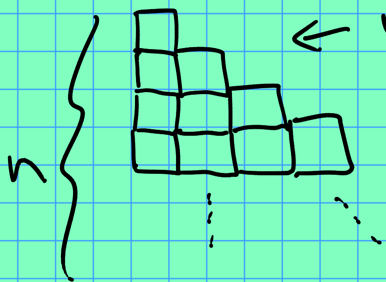
```
void growArray(int*& A, int size, int newSize)
{
        int* N = new int[newSize]; //1
        // we assume newSize > size...
        for (int i = 0; i < size; i++)   //2
            N[i] = A[i];

        delete[] A;   //3
        A = N;
}
```

Back to our question:

cost for $n$ consecutive push_back ?

Say we did size++ instead of size*=2 ?

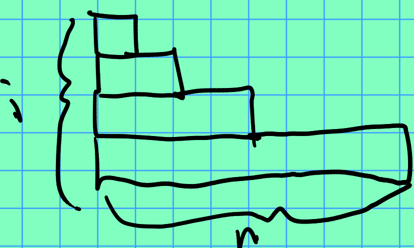


← work for growing array

So, cost $\geq \dfrac{n(n+1)}{2}$

$$\left( = \sum_{i=1}^{n} i \right)$$

Now what if we do size*=2 ?



$$\approx \sum_{i=0}^{\log n} 2^i =$$

$$\frac{(1-2)\left(1+2+2^2+2^3+\dots 2^k\right)}{(1-2)}$$

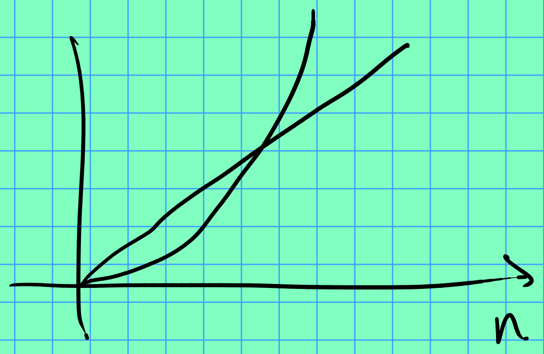
$$= \frac{(1-\cancel{2})+(\cancel{2}-\cancel{2^2})+(\cancel{2^2}-2^3)+\dots+\cancel{2^k}-2^{k+1}}{1-2}$$

$$= \frac{1-2^{k+1}}{1-2} = 2^{k+1}-1$$

But our $k \approx \log n$.

So, cost of reallocation (total, after $n$ push_back calls)

$$\approx 2^{\log_2 n+1} - 1 = 2n-1$$



n