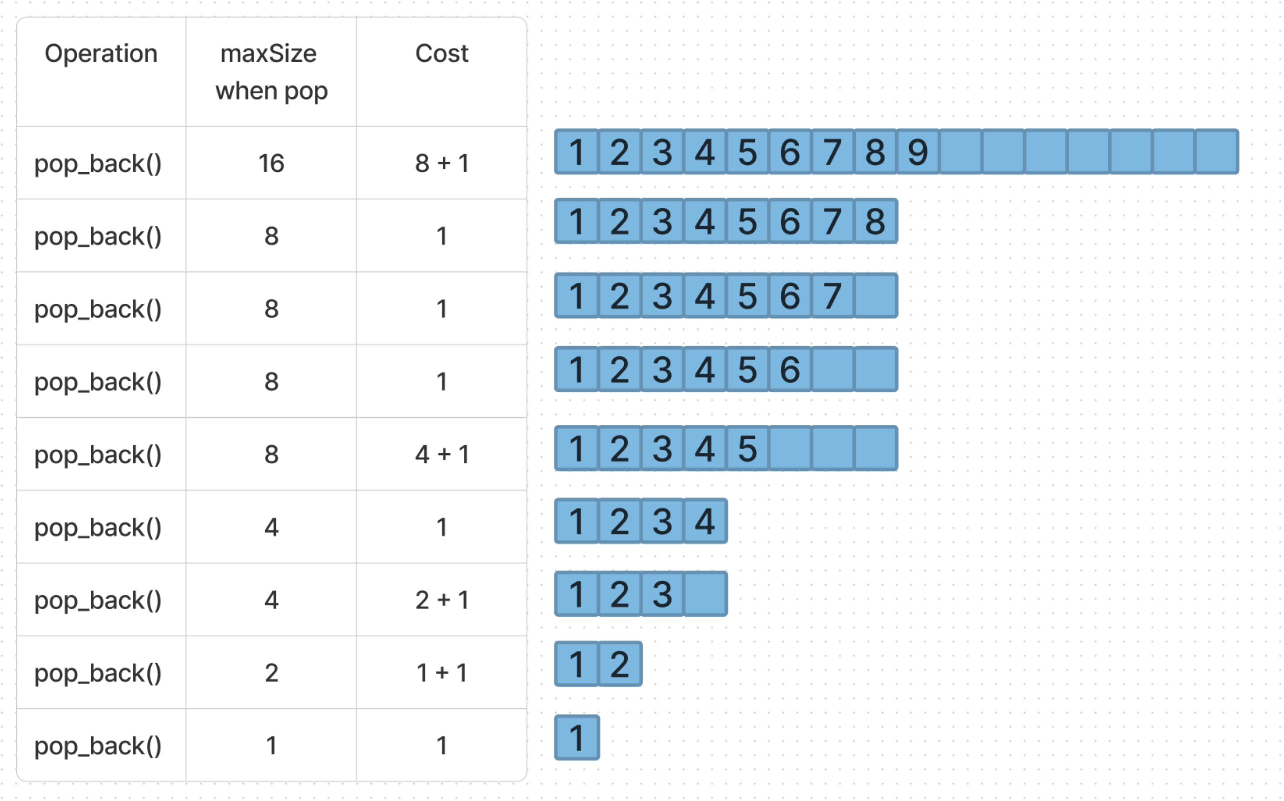


Operation	maxSize when pop	Cost
pop_back()	16	8 + 1
pop_back()	8	1
pop_back()	8	1
pop_back()	8	1
pop_back()	8	4 + 1
pop_back()	4	1
pop_back()	4	2 + 1
pop_back()	2	1 + 1
pop_back()	1	1

Amortized cost (C_i) = Actual cost(c_i) + change in potential

- **Slow Operation:**

  $maxSize_i = k$, $maxSize_{i-1} = 2*k$, $n_i = k$, $n_{i-1} = k+1$

  // $maxSize = 8$, $maxSize_{i-1} = 16$, $n_i = 8$, $n_{i-1} = 8+1$

  $C_i = c_i + \Phi(v_i) - \Phi(v_{i-1})$

  $= (k+1) + (maxSize_i - 2*n_i) – (maxSize_{i-1} - 2*n_{i-1})$

  $= (k+1) + (k – 2*k) – ((2*k) – (2*(k+1))$

  $= 3$

- **Fast Operation:**

  $maxSize_i = s$, $maxSize_{i-1} = s$, $n_i = k$, $n_{i-1} = k+1$

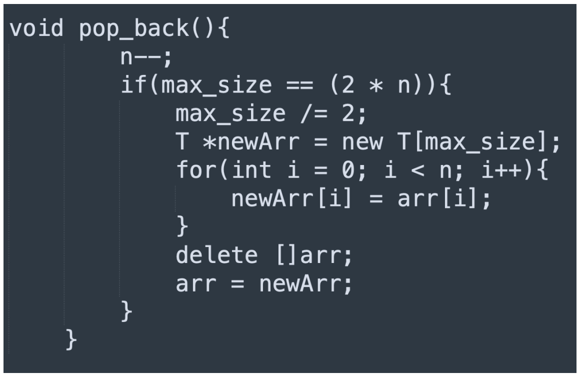
  // $maxSize = 8$, $maxSize_{i-1} = 8$, $n_i = 6$, $n_{i-1} = 7$

  $C_i = c_i + \Phi(v_i) - \Phi(v_{i-1})$

  $= 1 + (maxSize_i - 2*n_i) – (maxSize_{i-1} – 2*n_{i-1})$

  $= 1 + (s – 2*k) – (s-2*(k+1))$

  $= 3$



```
void pop_back(){
        n--;
        if(max_size == (2 * n)){
            max_size /= 2;
            T *newArr = new T[max_size];
            for(int i = 0; i < n; i++){
                newArr[i] = arr[i];
            }
            delete []arr;
            arr = newArr;
        }
    }
```