amortized O(1) for push operations:

for normal push operations time complexity of push operation is O(1).

push with grow: for pushing element with grow ,we create an array in heap with size equal ro (growth factor)*capacity and copy all the current elements of original array in new array and finaly delete original array. copying original array takes O(n) time complexity.

finaly pushing after grow:capacity has been updated to newcapacity . hence,extra space is available for inserting element . so push will take O(1) time complexity.

capacity doubls when size=capacity here capacity's 1024 initialy
when size is 1024,
1.adding num in array cost= 2(size=1024)+1(cost of insertion)
2.cost of adding nums upto when size=2048(growthf*capacity) will be 1 (since no copying occurs at moment)
now when size is 2048(growthf*size)
3.adding num in array cost=2(size=2048)+1
4.similar as point 2

#cost for normal operations=1
#cost for grow operation=2^i*(capacity=1024)(creating a new array) +2^(i-1) (copying prev elements)=2^(i+1)(for ith operation)

#amortized complexity=((time complexity(normal op's))+(time complexity(grow op's)))/n (using aggregate methode)

#total cost of n operations=(1+1+1...nterms)+(initial cap.=1024)*(2+4+8+....kterms)
amortized_cost=((1+1+1...nterms)+(initial cap.=1024)*(1+2+4+....kterms))/n

now, 2^k=n;(like dividing rod of n in two parts until a unit length remains)

k=log(n)/log2+1 ((2^(k-1)=n) for (2+4 +...) and 1 for 1st grow operation)

#cost of n operations=(n+(ci)*(2+4...+kterms))/n

amortized_cost<=(n+ci*(4*n-1))/n

amortized_cost<=(ci*4+1)(ci here is initial capacity which is 1024)

amortized_cost=amortized complexity (since both formulas given above are equivalent)

amortized complexity =(4*ci+1)=O(1)

amortized O(1) for pop operations:
cost of normal pop operation:O(1) (deleting top element takes constant time)
cost of shrinking: since it's multiplicative decrease (by factor of 2) when size<thsize(here capacity/4) also it refers to hysterisis.
(hysterisis is when you do a shrink operation after size has reduced below certain threshold , this creates zone for small flactuations around that threshold size which prevents frecuent shrnking operations and hence reduces cost)

using potentiel methode,
it takes shrink_ cost to shrink an array
shrink_=(curr_size)+(curr_capacity)/2
assuming curr_size=thsize
cost of shrink operation=curr_capacity/4+curr_capacity/2

for i operations,
amortized cost=(1+1+1...+n)+(capacity/4+capacity/2+capacity/8+capacity/4....kterms)
(exact no of terms for this part can't be determined with formula since whenever capacity shrinks below 1024 it's get resized to size equal to 1024 (it's minimum capacity of an array) but we can use above formula)
if(capacity<1024 we set capacity to 1024)
since it's already lesser than cost for pushing operation,
amortized cost<=(n+ci*(4*n))/n
amortized complexity=O(1).