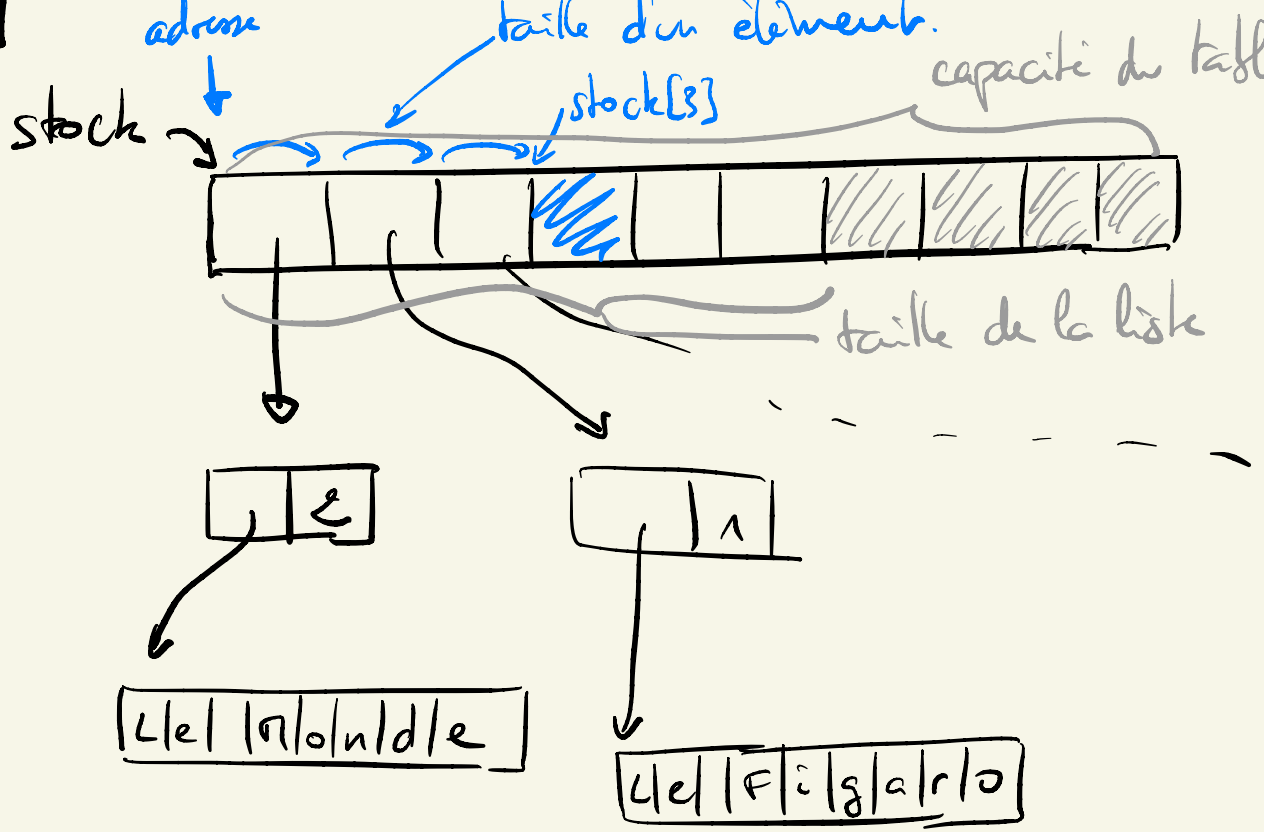
Q2



stock[i] : adresse + i × taille

calcul en $O(1)$

⇒ get / set : parcourt de la liste pour trouver le journal recherché

$O(n)$

si n est la longueur de la liste

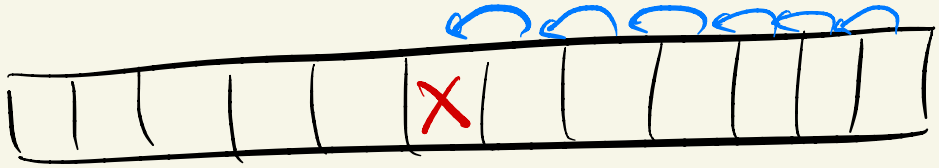
Q3

add : deux cas :

- la capacité du tableau est suffisante pour accueillir le nouvel élément $O(1)$
- le tableau est rempli au max de sa capacité : on alloue un nouveau tableau de capacité double, et on recopie les éléments dedans $O(n)$

$O(1)$ complexité "amortie"

Q4 **del** : il faut trouver l'élément : $O(n)$



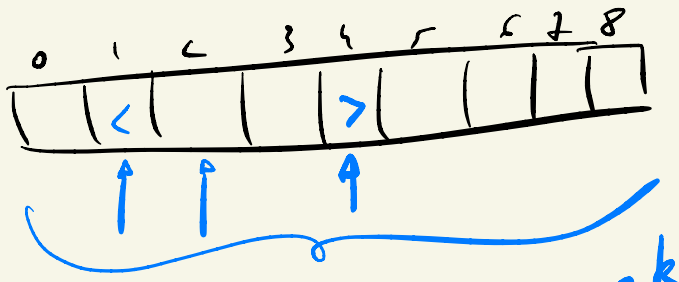
il faut aussi décaler tous les éléments situés après, d'un cran vers la gauche $O(n)$

Q5 Liste triée par ordre alphabétique ⇒ recherche dichotomique

get : $O(\log n)$

set : $O(\log n)$

add : $O(\log n)$ pour la recherche
$+ O(n)$ pour déplacer les éléments qui sont après
$\Rightarrow O(n)$

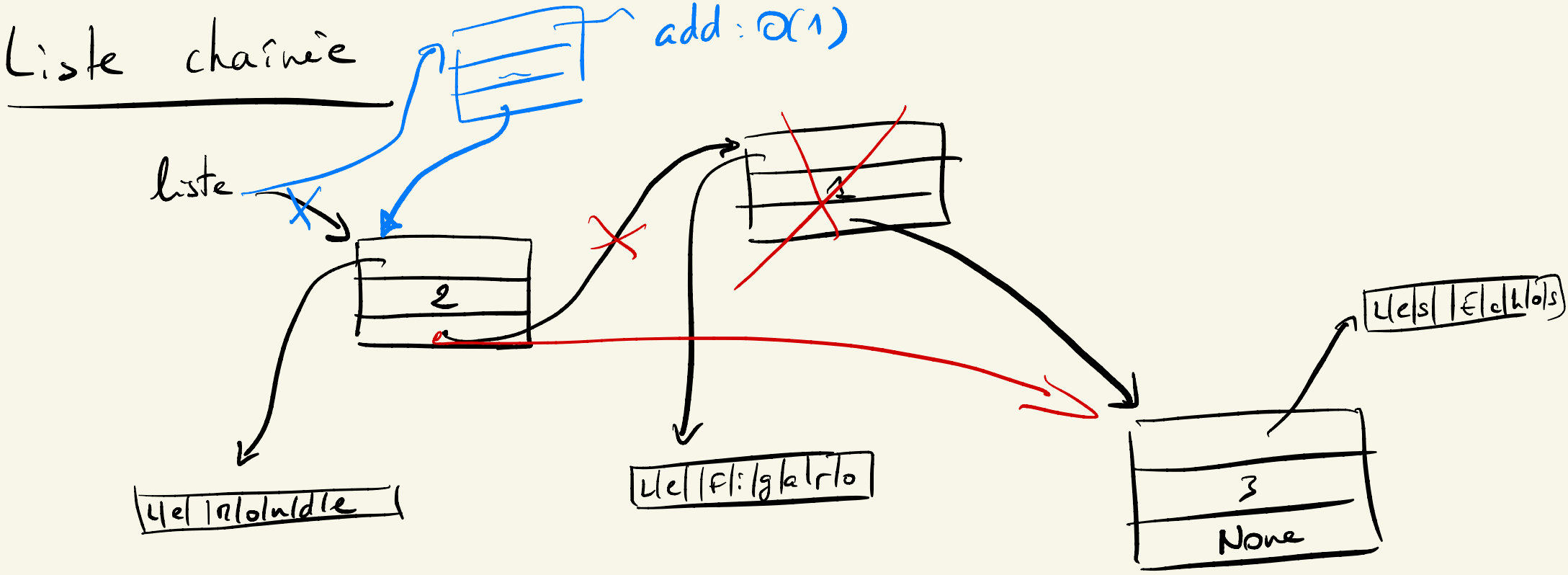


n cases $= 2^k$

$k \approx \log_2 n$

del : idem que add

Liste chaînée



add : $O(1)$

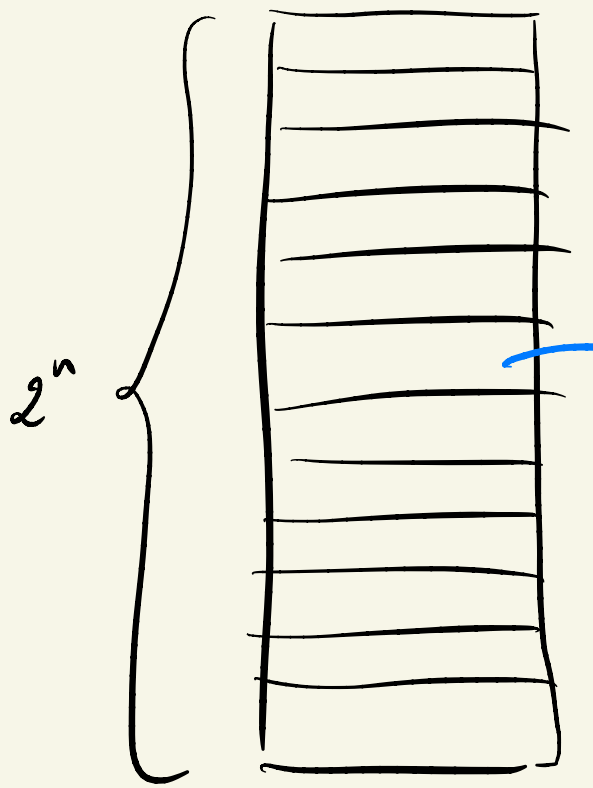
get et set : $O(n)$ car on parcourt la liste en suivant le chaînage

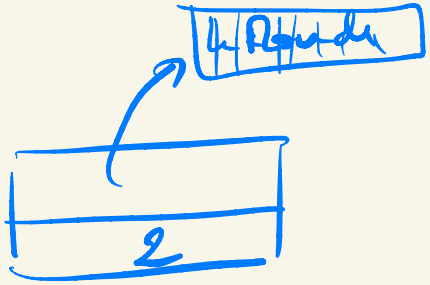
⚠ Si on fait le parcours en utilisant ieme()
on passe en $O(n^2)$ car ieme(i) est en $O(i)$

add : $O(1)$ car on ajoute le nouveau maillon comme 1er maillon

del : $O(n)$ pour trouver l'élément et ensuite $O(1)$ pour le supprimer (mise à jour de la référence vers le maillon suivant)

Dictionnaires python : tables de hachage





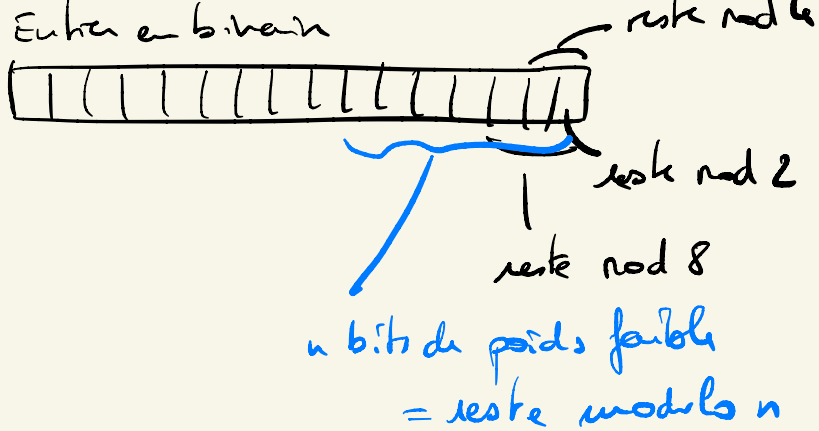
dico['la Rande']

↳ .hash() → entier $[2^n]$

↓

indice de l'entrée dans la table

Q.10 : pourquoi $N = 2^n$



Piles et tas

Représentation d'un arbre binaire dans un tableau :

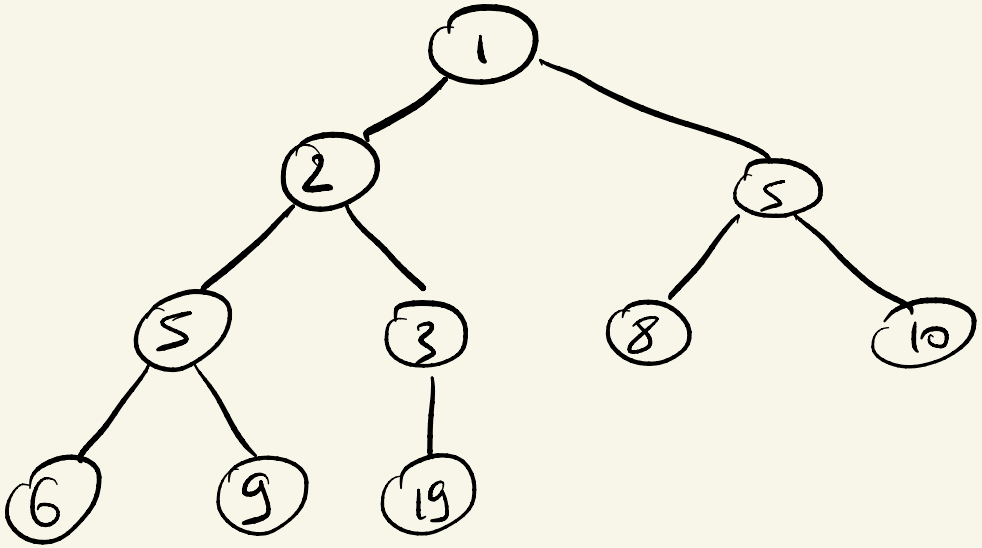
p : taille du tas

heap-pop : $\mathcal{O}(\log p)$

heap-push : $\mathcal{O}(\log p)$

pop : $\mathcal{O}(1)$

push : $\mathcal{O}(1)$



1	2	5	5	3	8	10	6	9	19	

racine : indice 0

fils gauche de i : $2i+1$ fils droit de i : $2i+2$

parent de i : $\left\lfloor \frac{i-1}{2} \right\rfloor$

Tri : n copies On prend n fois la pile de copie qui est au sommet du tas min, on pop la 1ère copie de la pile, et on remet la pile dans le tas : $2 \log p + 1$ En tout $\mathcal{O}(n \log p)$