$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ | $\sum_{i=1}^{n} i^2 = \frac{1}{6} n(n+1)(2n+1)$ | $\sum_{i=0}^{\infty} a^i = \frac{1}{1-a}$ *für* $0 < a < 1$ | *Teleskopsumme:* $\sum_{i=m}^{n} (a_{i+1} - a_i) = a_{n+1} - a_m$

Rekursion: Verfahren: rekursives Einsetzen, Vermutung aufstellen nach k-maligem Einsetzen, Beweis der Vermutung über Induktion nach k, Rückführung der Vermutung auf den Basisfall

$f = O(g) \Leftrightarrow$ Es gibt eine positive Konstante $c > 0$ und eine natürliche Zahl $n_0 \in \mathbb{N}$ so dass $\forall\, n \geq n_0$ gilt: $f(n) \leq c \cdot g(n)$ $f = o(g) \Leftrightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)}$

$\sum_{i=0}^{n} a^i = \begin{cases} \frac{a^{n+1}-1}{a-1}, & falls\ a \neq 1 \\ n+1, & sonst \end{cases}$

Indexverschiebung: $\sum_{i=m}^{n} a_i = \sum_{i=m-k}^{n-k} a_{i+k}$ $\forall\, k \in \mathbb{Z}$ *mit* $m - k \geq 0$

Rückwärtssummation: $\sum_{i=m}^{n} a_{n-1} = \sum_{i=0}^{n-m} a_i$

$a^{\log_a n} = n$ | $\log_a n = (\log_a b)(\log_a n)$

$f(i) = o(f_{i+1})$: $\log_a n \ll \sqrt{n} \ll n \ll n \log_a n \ll n^k \ll b^n \ll n!$

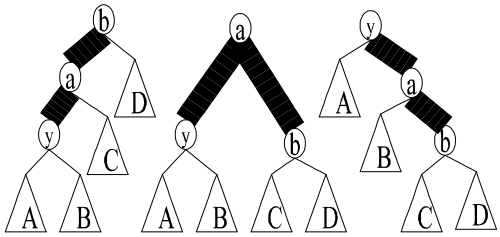
Zu lösen sei: $T(1) = c$, $T(n) = a \cdot T(\frac{n}{b}) + f(n)$ *mit* $c > 0$, $a \geq 1$, $b > 0$

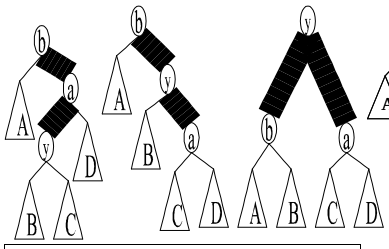
Kleiner Overhead: *Es sei* $f(n) = O(n^{\log_b a - \varepsilon})$ *für eine Konstante* $\varepsilon > 0 \Rightarrow T(n) = \Theta(n^{\log_b a})$

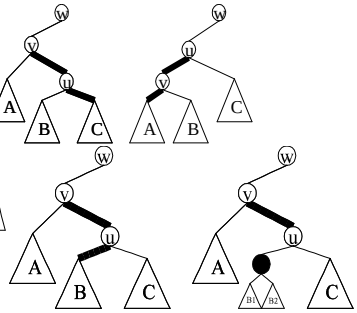
Mittlerer Overhead: *Es sei* $f(n) = \Theta(n^{\log_b a}) \Rightarrow \quad T(n) = \Theta(n^{\log_b a} \cdot \log_b n)$

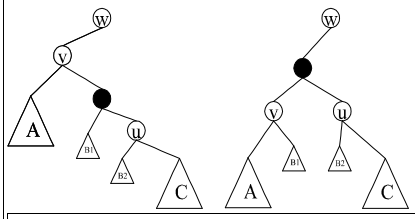
Großer Overhead: *Es sei* $f(n) = \Omega(n^{\log_b a + \varepsilon})$ *Konstante* $\varepsilon > 0$ *und a* $f(\frac{n}{b}) \leq \alpha f(n)$ *Konstante* $\alpha < 1$

$\Rightarrow T(n) = \Theta(f(n))$









Jeder Knoten der Tiefe <= t-2 hat genau 2 Kinder (Heap-Struktur)
Wenn ein Knoten der Tiefe t-1 weniger als 2 Kinder hat, dann haben alle Kinder rechts von ihm keine Kinder
Wenn ein Knoten nur ein Kind besitzt, dann ist es ein linkes Kind
Für jeden Knoten u und jedes Kind v gilt: p(u) >= p(v) (Heap-Ordnung)
Im Heap-Array A gilt für jeden Knoten u und dessen Kinder v,w: A[u]=p(u), A[2u]=p(v), A[2u+1]=p(w)

Tiefe = $\lfloor \log_2 n \rfloor$

alle Operationen O(log n), Build Heap O(n)

Heapsort O(n log n)
Insert repair_up: in freie Stelle einsetzen; mit Vätern tauschen, falls nötig –
Delete_max repair_down: entferne Wurzel setze letztes Element ein; vertausche mit größten Kindern

(Topologisches Sortieren: Datenstruktur Queue O(|V|+|E|))
Initialisiere Adjazenzliste; Initialisiere Verstoss-Array; Setze Zähler=0; Wiederhole solange bis Schlange leer: {Entferne a_i aus Schlange; Setze Reihenfolge[Zähler++]=i; Verstoss[b_i]-- aller Nachfolger b_j in der Adjazenzliste von a_i
Bäume - Implementierung:
Vater-Array (schnelle Vater- und Wurzelbestimmung)
Kind-Geschwister-Darstellung (Kindbestimmung über Lkind und dann Geschwister-Zeiger)

Binärbaumstruktur (nur für Binärbäume)
Suche: alle in O(n)
Präorder (Knoten v, dann linkes Kind, dann rechts Kind))
Postorder (Knoten v, dann linkestes Blatt im rechten Geschwisterbaum)
Inorder(T_links, Wurzel v, T_rechts)

$Tiefe \geq \log_2(n!) \geq \frac{n}{2} \log \frac{n}{2}$ Untere Schranke – Sortierbaum vergleichsbasiertes Sortieren ?(n log n)

Bubblesort: schnell, wenn maximaler Abstand zur endgültigen Position klein für jede Position
Selectionsort (nur n-1 Vertauschungen):
for(i=1; i<n;i++) A[i]=min {A[i], ..., A[n]}
Insertionsort (schnell bei wenigen Schlüsseln <= 30 und bei fast sortierten Daten):
for(i=1; i<n;i++) Fuege A[i+1] in die sortierte Folge A[1], ..., A[i] ein (verschiebe dabei die größeren um 1 nach rechts;

Quicksort
quicksort(links, rechts){ if (links < rechts){ p=pivot(links, rechts); i=partition(p, links, rechts); quicksort(links, i-1); quicksort(i+1, rechts)} }
partition(p, links, rechts){ A[p]=a; l=links-1; r=rechts; vertausche A[rechts] mit A[p]; Wiederhole bis l>=r: {l++ bis A[l]>=a; r-- bis A[r]<=a; Wenn l<r, dann vertausche A[l] mit A[r];} Vertausche A[rechts] mit A[l]; }
pivot O(n), partition O(rechts-links+1), worst-case $O(n^2)$, erwartete Laufzeit bei Pivot zufällig O(nlogn), Korrektheit Induktion über Eingabelänge L=rechts-links+1
Verbesserung: Stack, Stackhöhe $\log_2 n$
Auswahlproblem (k-kleinste Schlüssel): modifiziertes Quicksort O(n)
ausgeglichenes Mehrweg-mischen : for (i=0; i<=n/m; i++) { Lade (A[i*m+1], ..., A[(i+1)*m]) in den Hauptspeicher und sortiere; Das sortierte Array wird auf Band i mod b zurückgeschrieben}
nur $1 + \lceil \log_b \frac{n}{m} \rceil$ Zugriffe

Mergesort
mergesort(links, rechts){ if (rechts > links){ mitte=(links+rechts)/2; mergesort(links, mitte); mergesort(mitte+1, rechts); merge(links, mitte, rechts); } }
merge(links, mitte, rechts){ for (i=links; i<=mitte; i++) B[i]=A[i]
for (i=mitte+1; i<=rechts; i++) B[i]=A[rechts-i+mitte+1];
i=links; j=rechts; for (k=links; k<=rechts; k++) A[k]= (B[i]<B[j]) ? B[i++]: B[j--]; }
Sortieren auf Band ?(n logn), aber doppelte Engabearraygröße, „Postorderdurchlauf“
Distribution Counting
O(n+m) linear für m <= n mit m Universum der Zahlen
counting(){Zähle[m]; wo=0; for(i=o;i<m;i++) Zähle[i]=0; for(j=1;j<=n;j++)Zähle[A[j]]++; for(i=0:i<m;i++){ for(j=1; j<=Zähle[i]; j++) A[wo+j]=i; wo+=Zähle[i]} }
Radixsort
Ein Array von b leeren Queues wird angelegt; for(j=1;j<n;j++) füge A[i] in die Queue s(j,i) ein, s(j,i) ist die i-niedrigswertigste Ziffer von A[j] ist; // Verteilungsphase zähle=1; for(j=0;j<b;j++) solange Queue j nicht leer, entferne das Element am Kopf der Queue und weise es A[zähle] zu, zähle++; // Sam-melphase Körbe von unten entleeren ?
stabil O($\log_b$m(n+b))
Linearzeit, aber nicht für große Zahlen:
b <= n ? O(n $\log_n$m)
m = n^k ? O(n k)
b = n ? $O(n^2)$ m = n^n