GÜ 07

Amortisierte Laufzeitanalyse:

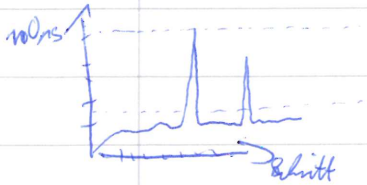
Motivation: Wir möchten die Kosten (in der Regel Anzahl Operationen) eines Algorithmus noch oben abschätzen.

↳ Problem: Diese können sich extrem unregelmäßig von Schritt zu Schritt verhalten, z.B. in vielen Schritten konstant und in wenigen logarithmisch oder linear.

⇒ Klassische Analyse potentiell sehr schwer, ist Algo ↑ etwa logarithmisch beschränkt oder doch linear? Vielleicht sogar in Schritt pro Schritt konstant?

Potentialfunktion: Wir betrachten das ganze Fortsetzungsverfahren. Einführung eines „Guthabens": Wir können einige „teure" Schritte über unser Guthaben ausgleichen, wenn wir dies in vorherigen Schritten ausgleichen.

Intuitiv: Zusätzlich zu den tatsächlichen Kosten eines Schritts können wir zusätzliche Schritte „bezahlen" obwohl wir diese gar nicht ausführen. Dies ist unser Potential, welches wir „aufladen". Wenn nun ein teurer Schritt durchgeführt ist, können wir von den tatsächlichen Kosten unser Guthaben abziehen! Damit kostet dieser Schritt nun deutlich weniger.



Potential darf nie negativ werden! -> Warum? $\Phi(D_i)$

$Cost_{amortized}(D_i) = Cost_{actual}(D_i) + \Phi(D_i) - \Phi(D_{i-1})$ D_i: innerer Zustand der Datenstruktur

$Cost_{amortized}(D) = \sum_{i=1}^{n} (Cost_{actual}(D_i) + \Phi(D_i) - \Phi(D_{i-1})) = Cost_{actual}(D) + \Phi(D_n) - \Phi(D_0)$

$Cost_{actual}(D) = Cost_{amortized}(D) - \Phi(D_n) + \Phi(D_0)$ ∈ Upper Bound für actual time!

Ansatz: Bei Verdoppelung: Kosten wie Kehr der neuen Dimension (d.h. nur Kosten für Allokation, vereinfachtes Modell)

A_i : Zustand des Arrays nach Schritt i

n_i : Anzahl bisher belegter Felder im Array vor Einfügen von i

m_i : Anzahl des insgesamt vorhandenen Arrayplatzes vor Einfügen von i

Vier mögliche Operationen: Suchen, Überschreiben, Einfügen ohne Erweiterung, Einfügen mit Erweiterung

Wir betrachten für alle Ops wie n_i und m_i sich verändern:

Suchen: $n_{i+1} = n_i$ $\quad m_{i+1} = m_i$

Überschreiben: $n_{i+1} = n_i$ $\quad m_{i+1} = m_i$

Einfügen ohne Erweiterung: $n_{i+1} = n_i + 1$ $\quad m_{i+1} = m_i$

" mit ": $n_{i+1} = n_i + 1$ $\quad m_{i+1} = 2m_i$

Wir erinnern uns an die Definition der amortisierten Kosten:

$a_i = t_i + \Phi(A_{i+1}) - \Phi(A_i)$

Wir müssen nun $\Phi(A_i)$ passend definieren!

↳ Was heißt passend? → Einfügen sollte amortisiert in konstanter Zeit möglich sein

$\Phi(A_i) := 4n_i - 2m_i$

Suchen, Überschreiben: $\Phi(A_i) := 4n_i - 2m_i$; $\Phi(A_{i+1}) = 4n_{i+1} - 2m_{i+1} = 4n_i - 2m_i$

$a_i = 1 + 4n_i - 2m_i - (4n_i - 2m_i) = 1 = O(1)$

Einfügen ohne Erweiterung: $\Phi(A_i) := 4n_i - 2m_i$ $\quad \Phi(A_{i+1}) = 4n_{i+1} - 2m_{i+1} = 4n_i + 4 - 2m_i$

$a_i = 1 + 4n_i + 4 - 2m_i - (4n_i - 2m_i) = 1 + 4 = 5 = O(1)$

Einfügen mit Erweiterung: $\Phi(A_i) := 4n_i - 2m_i$ $\quad \Phi(A_{i+1}) = 4n_{i+1} - 2m_{i+1} = 4n_i + 4 - 4m_i$

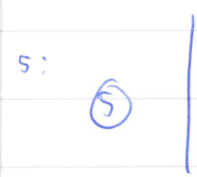
$a_i = 2m_i + 1 + 4n_i + 4 - 4m_i - (4n_i - 2m_i) = 5 = O(1)$

zig zag

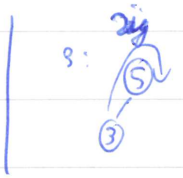
Splaytree

einfügen:

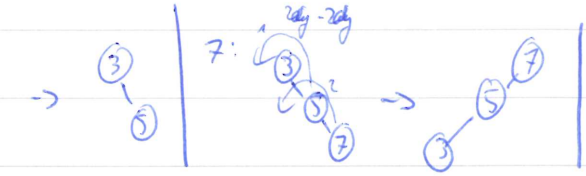
5:



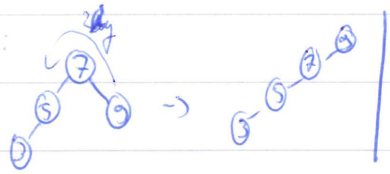
8:



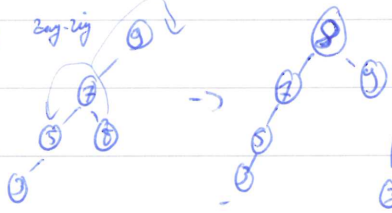
7:



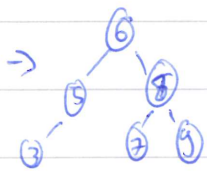
9:



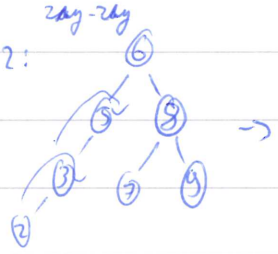
8:

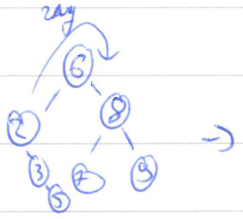


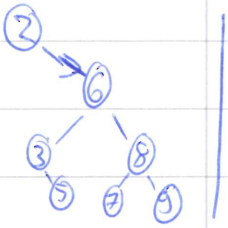
6:



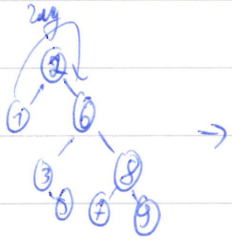
2:

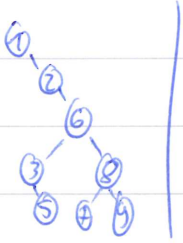




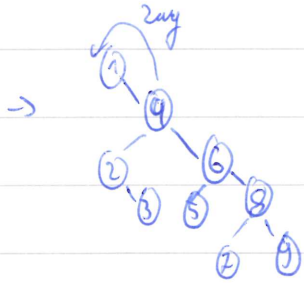


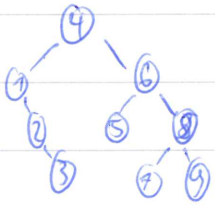
1:





4:





löschen

7: Splaye 7 in die Wurzel:

Splaye größtes Kind im l. Unterbaum (6)

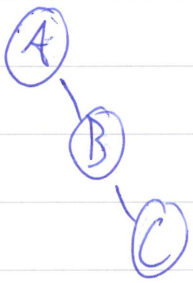
9: Splaye 9 in die Wurzel:

Splaye größtes Kind im l. Unterbaum (8)

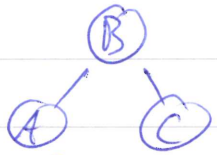
A $(33\frac{1}{3} + \varepsilon)$ $\varepsilon > 0$

B $(33\frac{1}{3} - \frac{\varepsilon}{2})$

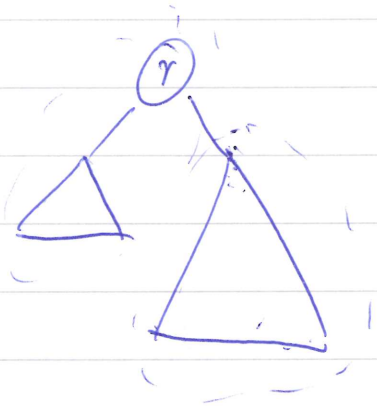
C $(33\frac{1}{3} - \frac{\varepsilon}{2})$



$200 - \frac{3}{2} * \text{epsilon} \approx 2$ Vergleiche



$166 + \frac{2}{3} + \frac{\varepsilon}{2} \approx 166$ Vergleiche



Motivation: Wir wollten die Kosten (in der Regel Operationen) eines Algorithmus nach oben abschätzen
↳ Problem: Diese Operationen können sich extrem unregelmäßig verhalten.
z.B. sehr häufig konstant viele Schritte aber manchmal linear viele.

Potentialfunktion: Wir betrachten das Ganze fortgesetzt Einzahlung eines „Guthabens". Wir können einige „teure" Schritte über unser Guthaben ausgleichen.

$$\underbrace{a_i(D_i)}_{\text{cost}_{\text{amortisiert}}(D_i)} = \underbrace{t_i(D_i)}_{\text{cost}_{\text{actual}}(D_i)} + \Phi(D_i) - \Phi(D_{i-1})$$

$$\underbrace{\text{cost}_{\text{amortisiert}}(D_n)}_{a_n(D_n)} = \sum_{i=1}^{n} \underbrace{\text{cost}_{\text{actual}}(D_i)}_{t_i(D_i)} + \Phi(D_i) - \Phi(D_{i-1})$$

$$= \text{cost}_{\text{actual}}(D) + \sum_{i=1}^{n} \Phi(D_i) - \Phi(D_{i-1})$$

$$= \underbrace{\text{cost}_{\text{actual}}(D)}_{t(D)} + \underbrace{\Phi(D_n)}_{\geq 0} - \underbrace{\Phi(D_0)}_{=0}$$

A_i : Zustand des Arrays vor Schritt i

n_i : Anzahl bisher belegter Felder in A_i vor Schritt i

m_i : Anzahl der insgesamt verfügbaren Arrayplätze vor Schritt i

$n_i = m_i \Rightarrow$ Arraygröße verdoppeln

Vier mögliche Operationen: Suchen, Überschreiben, Einfügen ohne Erweitern, Einfügen mit Erweitern

	n_{i+1}	m_{i+1}
Suchen	n_i	m_i
Überschreiben	n_i	m_i
Einfügen OE	$n_i + 1$	m_i
Einfügen ME	$n_i + 1$	$2m_i$

$$a_i = t_i + \Phi(A_{i+1}) - \Phi(A_i) \qquad m_i = 2n_i$$

$\Phi(A_i) = 2 + n \cdot 0 = 2 \;\Rightarrow\; \Phi(A_i) = 2 \cdot i$

$\Phi(A_i) := 4n_i - 2m_i$

Suchen, Überschreiben: $\Phi(A_i) = 4n_i - 2m_i \quad \Phi(A_{i+1}) = 4n_{i+1} - 2m_{i+1} = 4n_i - 2m_i$

$a_i = 1 + \Phi(A_{i+1}) - \Phi(A_i) = 1 = O(1)$

Einfügen OE:

$\Phi(A_i) = 4n_i - 2m_i \qquad \Phi(A_{i+1}) = 4n_{i+1} - 2m_{i+1} = 4(n_i + 1) - 2m_i$

$= 4n_i + 4 - 2m_i$

$a_i = 1 + \Phi(A_{i+1}) - \Phi(A_i) = 1 + 4n_i + 4 - 2m_i - (4n_i - 2m_i) = 5 = O(1)$

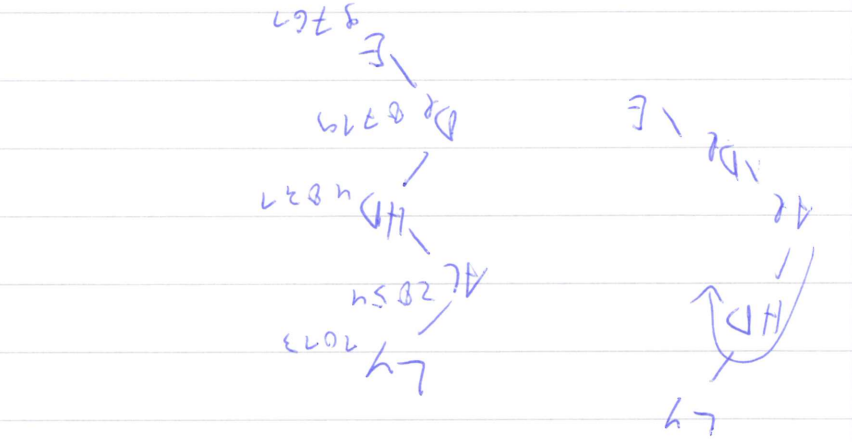
Einfügen ME:

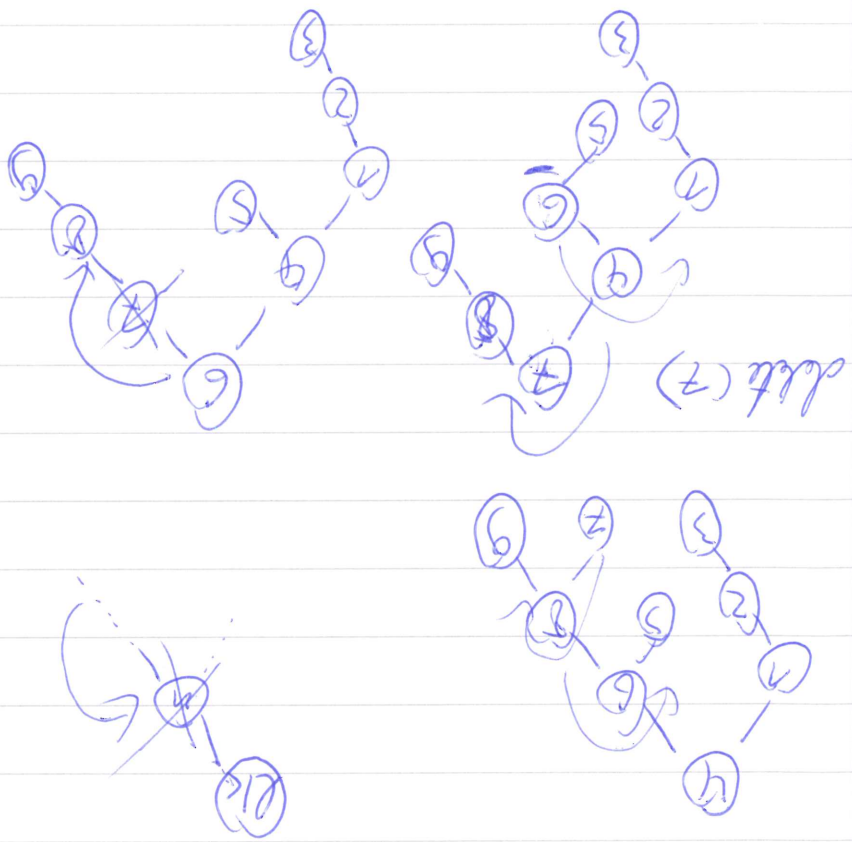
$\Phi(A_i) = 4n_i - 2m_i$ $\quad \Phi(A_{i+1}) = 4n_{i+1} - 2m_{i+1} = 4(n_i+1) - 2(2m_i)$

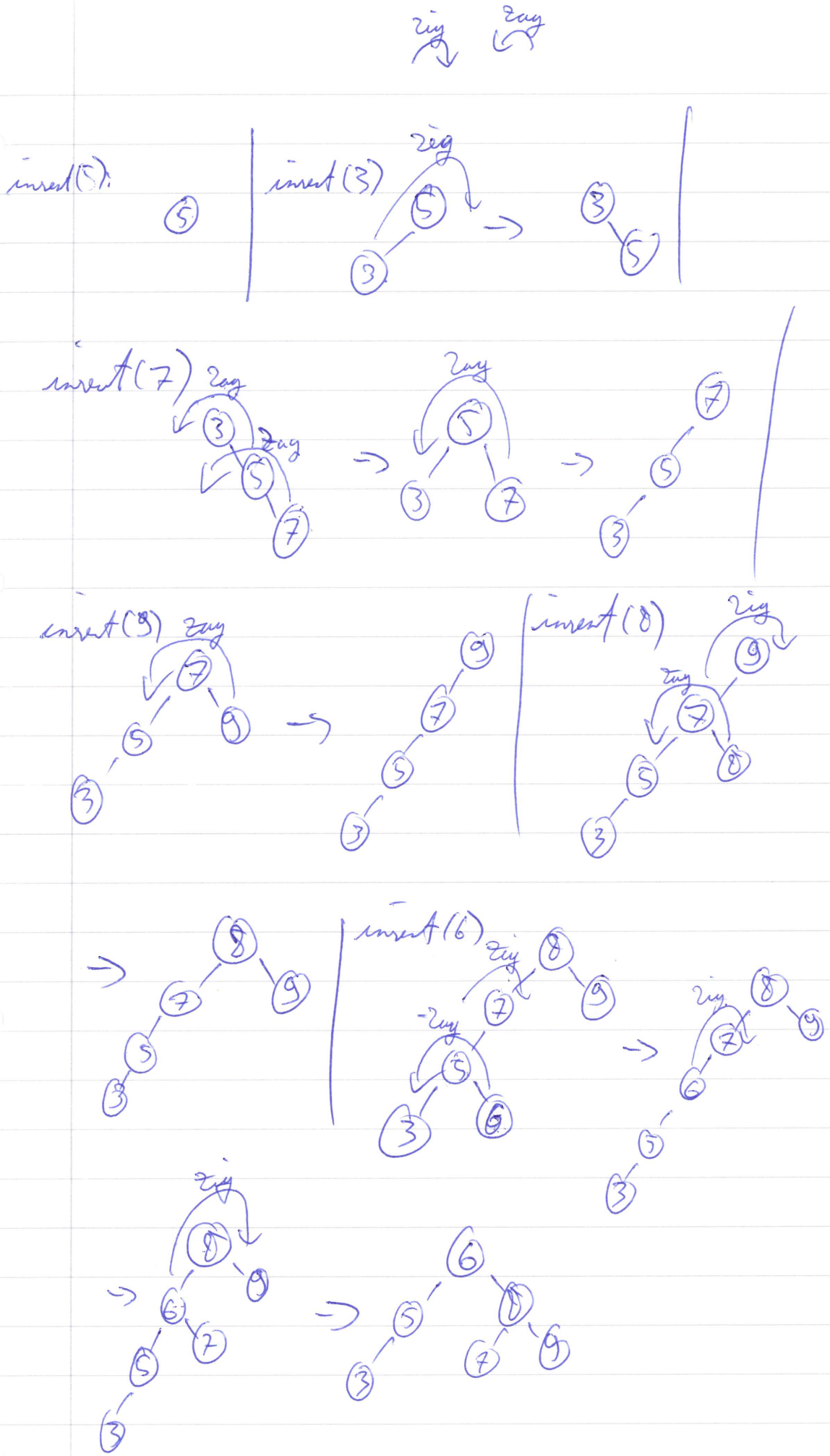
$= 4n_i + 4 - 4m_i$

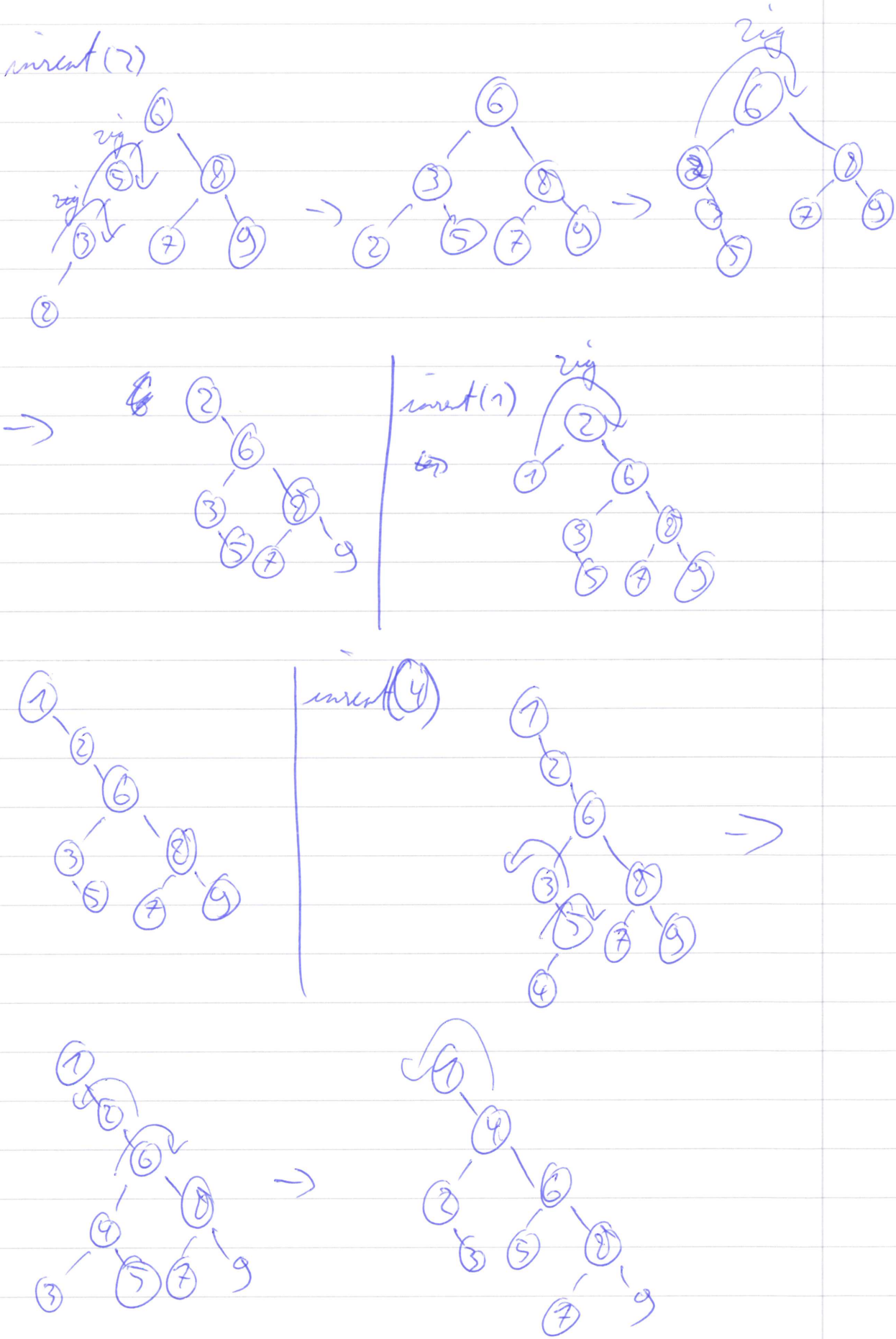
$a_i = \overbrace{1 + 2m_i}^{t_i} + \Phi(A_{i+1}) - \Phi(A_i)$

$= 1 + 2m_i + 4n_i + 4 - 4m_i - (4n_i - 2m_i) = 5 = O(1)$






zig
zag
insert(5)
insert(3)
zig
insert(7)
zag
zag
zag
insert(9)
zag
insert(8)
zig
zag
insert(6)
zig
zag
zig
zig


insert(7)
zig
zig
zig
insert(1)
insert(4)