

הגדרות

אלגוריתם - דרך שיטית וחד משמעית לביצוע של משימה מסוימת במספר סופי של צעדים

פסאדו קוד - תיאור מצומצם ולא רשמי לאלגוריתם של תוכנית מחשב

זמן ריצה - מספר פעולות היסוד המבוצעות על תוכנית כלשהי
למה פעולות יסוד ולא זמן? על מנת להתעלם מאספקטים טכנולוגים כמו סוגי מכונות

דוגמא: חיפוש בינארי מול חיפוש טרינרי

חיפוש בינארי	חיפוש טרינרי
$T(n) = \begin{cases} T\left(\frac{n}{2}\right) + 2, & n \geq 2 \\ 1, & n \leq 2 \end{cases}$ $T(n) = \dots = T\left(\frac{n}{2^k}\right) + 2 \cdot k = 2 \cdot \log_2 n$	$T(n) = \begin{cases} T\left(\frac{n}{3}\right) + 4, & n \geq 2 \\ 1, & n \leq 2 \end{cases}$ $T(n) = \dots = T\left(\frac{n}{3^k}\right) + 4 \cdot k = 4 \cdot \log_3 n$

$$2 \cdot \log_2 n = 2 \cdot \frac{\log_3 n}{\log_3 2} = 3.17 \cdot \log_3 n < 4 \cdot \log_3 n$$

אסימפטומטיקה - הערכה של קצב גידול של פונקציה

חסם עליון - נאמר ש- $f(n) \in O(g(n))$ אם:

$$\exists c > 0, n_0 \geq 0 : \forall n > n_0 \quad f(n) \leq c \cdot g(n)$$

חסם תחתון - נאמר ש- $f(n) \in \Omega(g(n))$ אם:

$$\exists c > 0, n_0 \geq 0 : \forall n > n_0 \quad f(n) \geq c \cdot g(n) \geq 0$$

חסם הדוק - נאמר ש- $f(n) \in \theta(g(n))$ אם:

$$\exists c_1, c_2 > 0, n_0 \geq 0 : \forall n > n_0 \quad c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

שם	סימון
אקספוננציאלי	$O(c^n)$
פולינומי	$O(n^c)$
	$O(n \cdot \log(n))$
לינארי	$O(n)$
	$O(\sqrt{n})$
לוגריתמי	$O(\log(n))$
	$O(\log(\log(n)))$
קבוע	$O(1)$
	$o\left(\frac{1}{n}\right)$

מיונים

הקדמה:

מיון מבוסס השוואות – מיון מבוסס השוואות מסדר אלמנטים במערך ע"י השוואה, בדרך כלל ע"י האופרטורים $\{\leq, \geq\}$



כל אלגוריתם המיון המבוסס על פעולות השוואה דורש לפחות $\Omega(n \cdot \log(n))$ פעולות השוואה **במקרה הגרוע ביותר**

הוכחה: עבור מיון n איברים יש צורך בעץ החלטה עם $n!$ עלים (עבור כל אחת מהפרמוטציות השונות), בעץ החלטה בגובה h יש לכל היותר 2^h עלים, ולכן:
 $2^h \geq n! \Rightarrow h \geq \log_2(n!) \in \Omega(n \cdot \log n)$

על מנת להשיג סיבוכיות מיון טובה יותר מ- $\Omega(n \cdot \log n)$ (ליניאריים) נוכל לאבד את הכלליות ולהתעסק במקרים פרטניים כמו **מספרים נמצאים בטווח חסום** (*Counting Sort*), **סדרה של מספרים הנמצאים בטווח חסום** (*Radix Sort*)

	מבוססי השוואות					לא מבוססי השוואות	
	<i>Bubble Sort</i>	<i>Selection Sort</i>	<i>Insertion Sort</i>	<i>Merge Sort</i>	<i>Quick Sort</i>	<i>Radix Sort</i>	<i>Counting Sort</i>
$T(n)$	$T(n) = (n-1) + \dots + 1 = n \cdot \frac{n-1}{2}$			$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n)$	$T(n) = T(k) + T(n-k-1) + O(n)$		
O	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot \log n)$	$O(n^2)$	$O(d \cdot (n+k))$	$O(n+k)$
Ω	$\Omega(n^2)$	$\Omega(n^2)$	$\Omega(n)$	$\Omega(n \cdot \log n)$	$\Omega(n \cdot \log n)$	$\Omega(d \cdot (n+k))$	$\Omega(n+k)$
Θ	$\Theta(n^2)$	$\Theta(n^2)$		$\Theta(n \cdot \log n)$	$\Theta(n \cdot \log n)$ בהסתברות גבוהה	$\Theta(d \cdot (n+k))$	$\Theta(n+k)$

מבני נתונים

מבנה נתונים	הגדרה	מוטיבציה	סיבוכיות הכנסה	סיבוכיות חיפוש	סיבוכיות מחיקה
מערך סטטי	מבנה שמורכב מאוסף של תאים סדרתיים בזיכרון בעל גודל קבוע	גודל המערך ידוע בזמן קומפילציה <ul style="list-style-type: none"> רוצים Random Access (גישה ישירה בזמן קבוע) 	$O(1)$	$O(n)$	$O(1)$
מערך דינמי	מבנה שמורכב מאוסף של תאים סדרתיים בזיכרון בעל גודל ניתן לשינוי	גודל המערך לא ידוע בזמן קומפילציה	$O(1)$ עד כדי $resize$ שלוקח $O(n)$	$O(n)$	$O(1)$
מערך דינמי ממויין	אוסף של תאים סדרתיים בזיכרון כך שכל איבר קטן מהאיבר הבא לפי אופרטור השוואה כלשהו $\{\leq, \geq, \dots\}$	נרצה לבצע חיפוש מהיר, כאשר יש יותר חיפושים מאשר הכנסות	$O(n)$ בגלל shifting של האיברים	$O(\log n)$	$O(n)$
רשימה מקושרת	<ul style="list-style-type: none"> לכל איבר יש ערך לכל איבר יש לכל היותר מצביע יחיד לאיבר הבא 	יותר הכנסות או אין צורך ב- Random Access , פחות אפקטיבי (זניח) מבחינת זיכרון עבור שמירה גם על גודל המצביע	$O(1)$	$O(n)$	$O(1)$
רשימה מקושרת ממוינת	<ul style="list-style-type: none"> לכל איבר יש ערך לכל איבר יש לכל היותר מצביע יחיד לאיבר הבא, שהוא קטן מהאיבר הבא לפי אופרטור השוואה כלשהו $\{\leq, \geq, \dots\}$ 	<ul style="list-style-type: none"> מימוש לתור עדיפויות היות ומחיקה מהראש ומהסוף היא פעולה קלה מחיקה של איברים זהים כאשר אין זיכרון לטעון ל-HashSet לדוגמא 	$O(n)$	$O(n)$	$O(n)$
מחסנית	<ul style="list-style-type: none"> מבנה נתונים אשר תומך בסדר פעולות היסטורי (האחרון הוא הראשון לצאת) 	LIFO	$O(1)$	-	$O(1)$
תור	מבנה נתונים אשר תומך בסדר פעולות היסטורי (האחרון הוא האחרון לצאת)	<ul style="list-style-type: none"> FIFO הערה: ניתן לממש תור בעזרת 2 מחסניות ומחסנית בעזרת 2 תורים (יש שקילות) 	$O(1)$	-	$O(1)$

$O(n)$	$O(n)$	$O(n)$		עץ בינארי (לכל קודקוד יש ערך ולכל קודקוד יש רשימה של לכל היותר 2 קודקודים) כך שעבור כל קודקוד בעל ערך x : 1. הערכים הקטנים מ- x נמצאים בתת עץ השמאלי 2. הערכים הגדולים מ- x נמצאים בתת עץ הימני	עץ חיפוש בינארי
$O(\log n)$	$O(\log n)$	$O(\log n)$	<ul style="list-style-type: none"> הכנסה, מחיקה, חיפוש יעיל חיפוש תווים, איברים סמוכים, מבנה נתונים ממויין 	<p>עץ חיפוש בינארי כך שכמות הרמות של לוגריתמית ביחס לכמות האיברים בעץ</p> <ul style="list-style-type: none"> AVL עץ אדום שחור <p><u>הוכחות עבור גובה עץ לוגריתמי</u></p> <p>היות ועץ AVL הוא עץ יותר מאוזן מעץ אדום-שחור, נעדיף להשתמש בו אם כמות החיפושים גדולה, אך אם מתבצעים הרבה שינויים כמות הכנסה או מחיקה, יתבצעו הרבה רוטציות ולכן נעדיף עץ אדום שחור שהוא יותר "סלחני"</p>	עץ חיפוש מאוזן
$O(\log n)$	עובר השורש $O(1)$ חיפוש כלשהו $O(n)$	$O(\log n)$ מכניסים לסוף, ואז עושים SwapUp עד לשורש	<ul style="list-style-type: none"> עץ עדיפות (במקום חיפוש נרצה עדיפות לפי אופרטור) <p>הערה: בנייה של עץ ערמה הוא $O(n)$ כי עושים Heapify כחצי מגודל המערך הוכחה</p>	<ul style="list-style-type: none"> כל האיברים בעץ ניתנים להשוואה ע"י אופרטור הערך של כל קודקוד קטן או שווה ביחס לאופרטור מהערך של בניו ממומש ע"י מערך כך שהבן השמאלי באינדקס $2i + 1$ והבן הימני באינדקס $2i + 2$ כאשר ראש הערמה באינדקס 0 <p>מבנה נתונים אשר תומך בעדיפות לפי אופרטור</p>	ערמה
<p>Direct Access Table: $O(1)$</p> <p>Open Addressing: $O(n)$</p> <p>Separate Channing: תלוי מימוש, מערך של רשימות מקושרות $O(\text{list length})$ בעוד שמערך של עצי חיפוש בינאריים $O(\log n)$</p>	<p>Direct Access Table: $O(1)$</p> <p>Open Addressing: $O(n)$</p> <p>Separate Channing: תלוי מימוש, מערך של רשימות מקושרות $O(\text{list length})$ בעוד שמערך של עצי חיפוש בינאריים $O(\log n)$</p>	<p>Direct Access Table: $O(1)$</p> <p>Open Addressing: $O(n)$</p> <p>Separate Channing: תלוי מימוש, מערך של רשימות מקושרות $O(\text{list length})$ בעוד שמערך של עצי חיפוש בינאריים $O(\log n)$</p>	<p>על מנת לאפשר חיפוש, הוספה ומחיקה בסיבוכיות קבועה, ניתן להשתמש ב-Direct Access Table מנת לקבל סיבוכיות זו ע"י $O(U)$ מקום כאשר U זה מרחב המפתחות האפשריים, או לצמצם את הזיכרון ולפתור בעיות התנגשות (Collision)</p> <p>אך, משלמים על זה בכך שמבנים דינמיים הם יעילים מבחינת זיכרון, בעוד שטבלת גיבוב היא בזבזנית, בנוסף אם אנחנו רוצים להתעסק עם מידע ממויין, מציאת איברים סמוכים, חיפוש תווים,</p>	<ul style="list-style-type: none"> לכל ערך יש מפתח בעל פונקציית גיבוב <p>פונקציית גיבוב טובה:</p> <ol style="list-style-type: none"> כמות הפלטים קטנה מכמות הקלטים האפשריים כמות התנגשויות צריכה להיות קטנה ככל שאפשר קלה לחישוב דטרמיניסטית תשתמש בכל המידע שיש למפתח מפזרת אחיד חד כיוונית לעיתים <ul style="list-style-type: none"> פותר התנגשויות 	טבלת גיבוב

	<p>בינאריים $O(\log n)$</p>		<p>להציג מבנה נתונים ממויין אז נעדיף להשתמש בעצים לדוגמא</p>	<p>לדוגמא:</p> <ul style="list-style-type: none"> • Open Addressing (מציאת מקום חדש לאובייקט) • Separate Chaining (מערך של רשימות מקשורות, עצי חיפוש, וכו') 	
--	---	--	--	--	--