Лекція 2. Аналіз алгоритмів

Глибовець А.М.

Вступ

- Сьогодні ми поговоримо про:
  - спостереження
  - математичні моделі
  - класифікацію за порядком зростання
  - теорію алгоритмів
  - пам’ять

Чарльз Беббідж

- «Як тільки Аналітична Машина буде створена, вона буде спрямовувати майбутній розвиток науки. Кожний раз коли буде потрібно отримати результат за її допомоги буде ставати питання який напрямок обрахунків, що проводяться машиною приведуть нас якомога швидше до результату» – Чарльз Беббідж 1864.
- В 1834 році Ч. Беббідж почав роботу над створенням програмованої обчислювальної машини, яку він назвав аналітичною.
- http://ru.wikipedia.org/wiki/%D0%91%D1%8D%D0%B1%D0%B1%D0%B8%D0%B4%D0%B6,_%D0%A7%D0%B0%D1%80%D0%BB%D1%8C%D0%B7
-

Running time

- За Беббіджем, час роботи вашого алгоритму вимірювався в тому, скільки разів ви маєте прокрутити ручку Аналітичної машини.

- Що змінилося зараз?
- Ми отримали електронну ручку, але все одно сильно залежимо від того, скільки разів ми маємо повторити дискретні операції

Причини аналізувати алгоритми

- Ми аналізуємо алгоритми що б:
  - Оцінити продуктивність
  - Порівняти алгоритми
  - Надати гарантії обчислюваності/виконуваності
  - Зрозуміти теоретичні основи
- З практичної точки зору:
  - ми хочемо усунути помилки продуктивності

Дискретне перетворення Фур'є

- Дискретне перетворення Фур'є (ДПФ, Discrete Fourier Transform) - це математична процедура, що використовується для визначення гармонічного, або частотного, складу дискретних сигналів.
- ДПФ є однією з найбільш розповсюджених і потужних процедур цифрової обробки сигналів.
- ДПФ дозволяє аналізувати, перетворювати і синтезувати сигнали такими способами, які неможливі при неперервній (аналоговій) обробці.
  - Самий простий алгоритм (brute force) потребує N^2 кроків
  - Алгоритм швидкого перетворення Фур’є (FFT) використовує N logN кроків. (був винайдений Гаусом ще в 1805 році)

Проблема

- Основне питання, що ставить собі програміст – чи зможе моя програма вирішити поставлену задачу на великих вхідних даних.
  - Чому моя програма така повільна?
  - Чому моїй програмі не вистачає оперативної пам’яті?
- Кнут в 1970 році сказав, що ми можемо використовувати науковий підхід до розуміння продуктивності програми.

Науковий підхід, що застосовується для аналізу алгоритмів

- Науковий підхід:
  - спостереження, якихось характеристик з реального світу, зазвичай на основі точних вимірювань
  - пропозиція гіпотетичної моделі, що узгоджується з спостереженнями
  - пророкування подій на основі запропонованої моделі
  - перевірка передбачень за допомогою подальших спостережень
  - обгрунтування за допомогою повторення процесу, поки гіпотеза і спостереження не співпадуть

Принципи наукового підходу

- Експерименти мають бути **відтворюємими**, що б інші могли впевнитися в вірності моделі, самостійно перевіривши гіпотезу.
- Гіпотези мають бути **фальсифікуємими**, що б можна було точно знати, коли гіпотеза не вірна.
- Висловлювання, що приписується Ейнштейну:
  - *Жоден об'єм експериментальних досліджень не може довести, що я правий, але всього один експеримент може довести, що я помиляюся.*

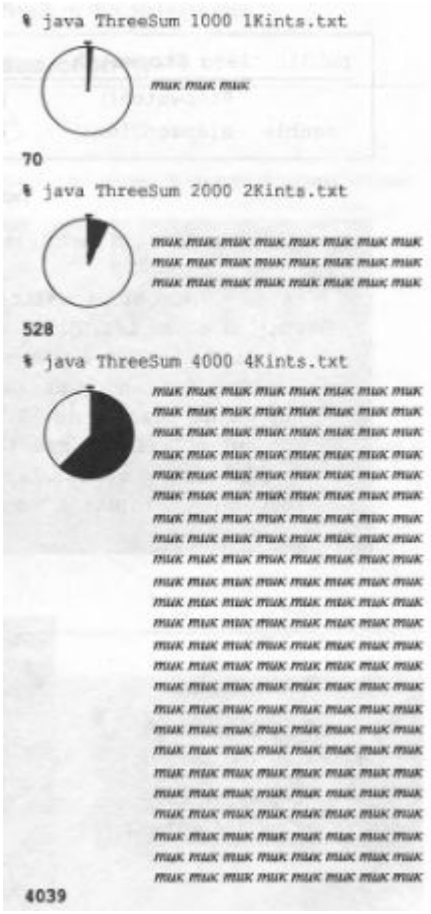
Спостереження

- Почнемо з спостереження.
- 3-Sum.
  - Дано N різних цілих чисел, скільки трійок в сумі дають 0?
    - На вхід ми отримали числа:
      - 30, -40, -20, -10, 40, 0, 10, 5
    - Відповідь:
      - 30, -40, 10
      - 30, -20, -10
      - -40, 40, 0
      - -10, 0, 10
  - Ця проблема має зв’язок з обчислювальною геометрією
  - Розглянемо розв’язання цієї проблеми
  - Перша спроба - TreeSumBF

Вимірювання часу роботи

- Як виміряти час роботи програми?
- Вручну.

http://standart-m.com.ua/userfiles/image/stopwatch1.jpg



Вимірювання часу роботи

- Як виміряти час роботи програми?
- Автоматично.
- Ми можемо скористатися класом Stopwatch()
  - **int[] a = In.~~readInts(*testFile*);~~**
  - Stopwatch stopwatch = **new Stopwatch();**
  - System.*out.println(count(a));*
  - **double time = stopwatch.elapsedTime();**
  - System.*out.println(time);*

Емпіричний аналіз

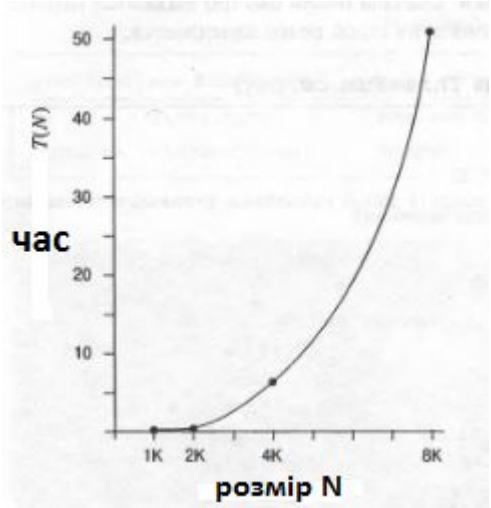
- Ми можемо запустити програму на різних об’ємах даних і оцінити витрачений час.
- Давайте спробуємо запустити з більшими об’ємами і подивимося на результат.
  - 8ints.txt
    - 4
    - Витрачений час 0.0
  - 1Kints.txt
    - 70
    - Витрачений час 0.654
  - 2Kints.txt
    - 528
    - Витрачений час 5.133
  - 4Kints.txt
    - 4039
    - Витрачений час 41.941
  - 8Kints.txt
    - 32074
    - Витрачений час 330.783

Емпіричний аналіз

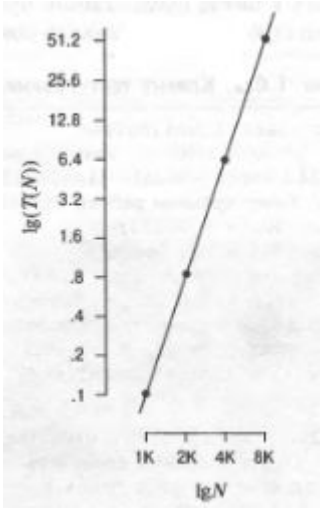
N	Час (секунди)
250	0.0
500	0.0
1000	0.1
2000	0.8
4000	6.4
8000	51.1
16000	?

Аналіз даних

- Зобразимо графічно залежність T(N) від N



Аналіз даних



- Намалюємо log-log графік T(N) від N
- ми отримали лінію з нахилом 3
  - рівняння такої прямої має вигляд:
    - lg(T(N))=3lgN+lga, де a константа
  - а це еквівалентно
    - $T(N) = aN^b = aN^3$
  - Таким чином ми отримали вираз часу виконання у вигляді функції від об'єму вхідних даних.
  - Можна взяти одну точку наших даних для визначення a
    - Наприклад:
      - $T(8000) = 51{,}1 = a*8000^3$, звідки $a=9{,}98*10^{-11}$
  - Тепер ми можемо використовувати формулу
    - $T(N) = 9{,}98 * 10^{-11} N^3$
  - для пророкування часу виконання для великих N

Аналіз даних

- Тепер ми отримали гіпотезу
  - на основі гіпотези ми можемо спрогнозувати дані
  - після чого провести серію експериментів і визначити чи співпадають реальні дані і дані за гіпотезою

Аналіз даних

- Незалежні чинники
  - Алгоритм
  - Вхідні дані
- визначають значення b в степені.
- Чинники залежні від системи
  - апаратне забезпечення
  - програмне забезпечення
  - система
- разом з незалежними чинниками впливають на значення константи a
- **Погані новини.** Важко отримати точні вимірювання.
- **Хороші новини.** Набагато простіше і дешевше, ніж в інших підходах.

Математична модель

- Д. Кнут – «незважаючи на складність, в принципі можливо побудувати математичну модель, що описує час виконання будь якої програми»
- Загальний час виконання програми залежить від:
  - вартості виконання кожного оператора
    - властивість комп'ютера, компілятора і операційної системи
  - частота виконання кожного оператора
    - властивість програми і вхідних даних

Математична модель
Вартість базових операцій

опреація	приклад	наносекунди
integer додавання	a+b	2.1
integer множення	a*b	2.4
integer ділення	a/b	5.4
float додавання	a+b	4.6
float множення	a*b	4.2
float ділення	a/b	13.5
сінус	Math.sin(theta)	91.3
Арктангенс	Math.atan2(y,x)	129.0
…	…	…

Математична модель
Вартість базових операцій

опреація	приклад	наносекунди
оголошення змінної	int a	c_1
привласнення	a=b	c_2
порівняння integer	a<b	c_3
досуп до елемента масиву	a[i]	c_4
довжина масиву	a.length	c_5
оголошення одновимірного масиву	new int[N]	c_6N
оголошення двувимірного масиву	new int[N][N]	c_7N^2
довжина стрічки	s.length()	c_8
знайдення підстрічки	s.substring(N/2,N)	c_9
об’єднання стрічок	s+t	$c_{10}N$

Математична модель

- Скільки інструкцій буде виконано в залежності від N?
  - int count = 0;
  - for (int i =0; i<N; i++)
    - if (a[i] == 0)
      - count++;

- Відповідь:
  - оголошення змінних – 2
  - привласнення значень -2
  - оператор порівняння – N
  - порівняння рівності N
  - доступ до елементів масиву N
  - збільшення на одиницю – від N до 2N

Математична модель

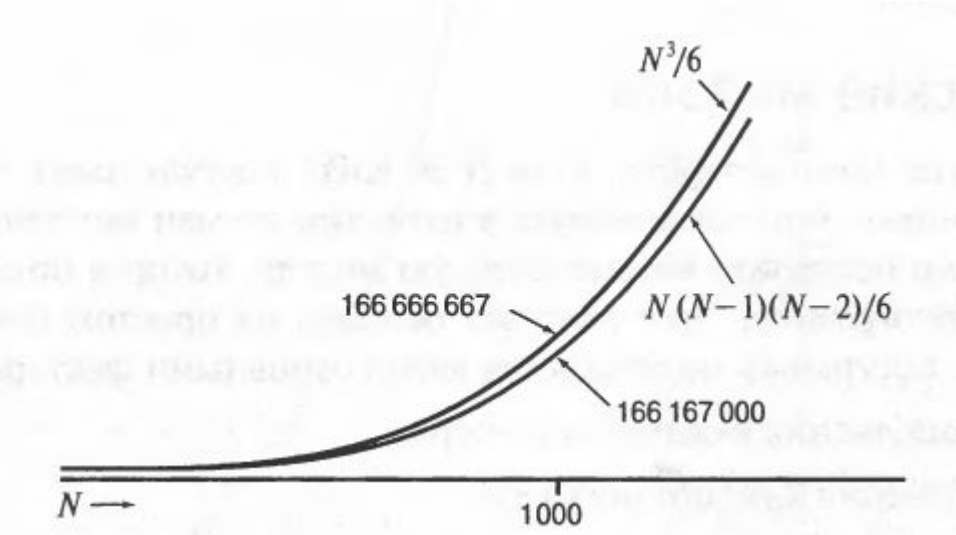
- Скільки інструкцій буде виконано в залежності від N?
  - int count = 0;
  - for (int i =0; i<N; i++)
    - for (int j =i+1; j<N; j++)
      - if (a[i]+ a[j] == 0)
        - count++;
- Відповідь
  - оголошення змінних – N+2
  - привласнення значень – N+2
  - оператор порівняння – ½(N+1)(N+2)
  - перевірка рівності – ½N(N-1)
  - доступ до елементів масиву - N(N-1)
  - збільшення на одиницю – від 1/2N(N-1) до N(N-1)

Математична модель

- «Дуже зручно мати міру об’єму робіт, навіть якщо вона буде дуже сира. Ми можемо підрахувати, скільки разів різні елементарні операції застосовуються в усьому процесі, а потім дати їм різної ваги.» - Алан Тюринг (1947).

Математична модель

- Замість того, що б обраховувати прискіпливо всі операції ми можемо ігнорувати відносно малі операції і таким чином спрощувати математичні формули.
- Це дозволяє нам працювати з апроксимацією.



Математична модель
Приклади апроксимації

Функція	Апроксимація старшим членом	Порядок зростання
$N^3/6 - N^2/2 + N/3$		

Ми будемо використовувати запис $\sim f(N)$, для позначення будь якої функції, що будучи поділеною на $f(N)$, наближається до 1 при збільшенні N.

Математична модель

- Скільки операцій доступу до масиву в наступному коді?
  - int count = 0;
  - for (int i =0; i<N; i++)
    - for (int j =i+1; j<N; j++)
      - if (a[i]+ a[j] == 0)
        - count++;
- Відповідь: $\sim N^2$

Математична модель

- Скільки операцій доступу до масиву в наступному коді?
  - int count = 0;
  - for (int i =0; i<N; i++)
    - for (int j =i+1; j<N; j++)
      - for (int k =j+1; k<N; k++)
      - if (a[i]+ a[j] +a[k]== 0)
        - count++;
- Відповідь:
  - $3 * \frac{N(N-1)(N-2)}{3!} = \sim \frac{1}{2} N^3$
- Як ви бачите ми не обраховуємо всі операції, ми беремо найдорожчі.

Математична модель

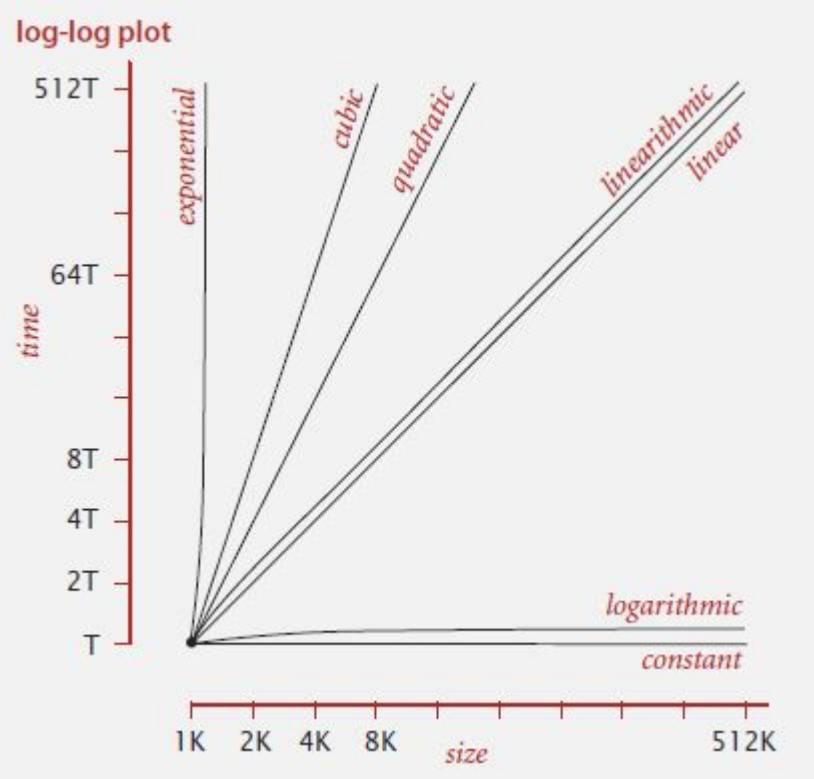
- Як оцінити дискретну суму?
- Замінити суму інтегралом і провести обрахунки.
- Приклад 1:
  - 1+2+…+N $\sum_{i=1}^{N} i \sim \int_{x=1}^{N} x dx \sim \frac{1}{2} N^2$
  - 1+ ½ + 1/3 +… +1/N $\sum_{i=1}^{N} \frac{1}{i} \sim \int_{x=1}^{N} \frac{1}{x} dx \sim lnN$
  - $\sum_{i=1}^{N} \sum_{j=i}^{N} \sum_{k=j}^{N} 1 \sim \int_{x=1}^{N} \int_{y=x}^{N} \int_{z=y}^{N} dz\, dy\, dx \sim \frac{1}{6} N^3$

Математична модель

- В принципі, математична модель можлива.
- На практиці:
  - формули можуть бути дуже складними
  - дуже складні обрахування потрібні
  - дуже точні моделі мало коли потрібні
- Ми будемо використовувати приблизні моделі.

Класифікація за порядком зростання

- Гарні новини – існує досить мало функцій, що описують порядок зростання.
  - $1, logN, N, NlogN, N^2, N^3, 2^N$



Класифікація за порядком зростання

Порядок зростаня	назва	Приклад коду	опис	Приклад використання
1	константа	a=b+c	вираз	додавання двох чисел

logN	логарифмічний	while(N>1){
		N=N/2;…
		}
			ділення навпіл	бінарний пошук

N	лінійний	for(int i=0;i<N; i++) {…}	цикл	знайти максимальне число
NlogN	лінійно-логарифмічний	пізніше	розділяй та пануй	сортування злиттям

	квадратична	for(int i=0;i<N; i++)
		for(int j=0;j<N; i++)
		{…}
			подвійний цикл	перевірка всіх пар

	кубічна	for(int i=0;i<N; i++)
		for(int j=0;j<N; j++)
		for(int k=0;k<N; k++)
		{…}
			потрійний цикл	перевірка всіх трійок

	експотенційна	пізніше	вичерпний пошук	перевірка всіх підмножин

Класифікація за порядком зростання

Порядок зростання	розмір проблеми, що розв'язуються за хвилини			
	1970-ті	1980-ті	1990-ті	2000-ні
1	будь-яка	будь-яка	будь-яка	будь-яка
logN	будь-яка	будь-яка	будь-яка	будь-яка
N	мільйони	десятки мільйонів	сотні мільйонів	більйони
NlogN	сотні тисяч	мільйони	мільйони	сотні мільйонів
	сотні	тисячі	тисячі	десятки тисяч
	сотні	сотні	тисячі	тисячі
	20	20	20	30

Бінарний пошук

- Дано відсортований масив і ключ, потрібно знайти індекс ключа в масиві.
- Бінарний пошук:
  - порівнюємо ключ з центральним елементом
  - якщо центральний елемент більше ключа, йдемо наліво
  - якщо менше, йдемо направо
  - рівний – знайшли відповідь.

- Як знайти 34?

0	1	2	3	4	5	6	7	8	9	10	11	12	13
6	13	15	24	35	42	50	51	54	62	74	84	95	96

Бінарний пошук

- Перший алгоритм бінарного пошуку опублікований в 1964 році
- Перший алгоритм без помилок – в 1992 році
- Помилки в Arrays.binarySearch() знайдені в 2006 році.
- Подивимося на реалізацію BinarySearch
- Твердження.
  - Бінарний пошук використовує 1+lgN порівнянь ключа і елементів масиву розміру N

Бінарний пошук: математичний аналіз

- $T(N)$ = кількість операцій порівняння в відсортованій підмножині розміру <=N
- $T(N) \leq T\left(\frac{N}{2}\right) + 1$ для $N > 1$,
- ми ділимо проблему навпіл і можемо реалізувати порівняння з однією операцією
- $T(N) \leq T\left(\frac{N}{2}\right) + 1$
- $\leq T\left(\frac{N}{4}\right) + 1 + 1$
- $\leq T\left(\frac{N}{8}\right) + 1 + 1 + 1$
- ...
- $\leq T\left(\frac{N}{N}\right) + 1 + 1 + \cdots + 1$
- $= 1 + lgN$

$N^2 \log N$ алгоритм для 3-суми

- Алгоритм базований на сортуванні для 3-суми
  - Відсортувати N чисел
  - Для кожної пари чисел a[i] і a[j] ми робимо бінарний пошук елемента –(a[i]+a[j])
- Порядок зростання $N^2 \log N$

Порівняння

N	час в секундах
1000	0.1
2000	0.8
4000	6.4
8000	51.1

brute-force

N	час в секундах
1000	0.14
2000	0.18
4000	0.34
8000	0.96
16000	3.67
32000	14.88
64000	59.16

sorting-based

Аналіз

- В реальності наші приклади набагато складніші ніж ті, що ми розглядали.
- І складність нашого алгоритму залежить від вхідних даних.
- Тому ми можемо оцінити **найкращий випадок**,
  - самий простий випадок вхідних даних
- та оцінити **найгірший випадок** (верхня межа вартості),
  - самий складний варіант вхідних даних
  - отримаємо гарантію того, що гірше вже не буде
- Після цього ми можемо отримати **середню складність**. Очікувані витрати для випадкових вхідних даних.

Аналіз

Brute-force 3 сума	
Найкращий	
Середній	
Найгірший	

Бінарний пошук	
Найкращий	
Середній	
Найгірший	

Теорія алгоритмів

- Основними цілями теорії алгоритмів є:
  - визначити «складність» проблеми
  - запропонувати «оптимальний» алгоритм

- Оптимальний алгоритм:
  - Має гарантовану продуктивність для будь яких вхідних даних
  - Не існує алгоритму, що може гарантувати кращу продуктивність

Загально прийняті позначення в теорії алгоритмів

Позначення	опис	приклад	Скорочення для	Використовується для
Велика Тета	Асимптотичний порядок зростання			Класифікації алгоритмів
Велике О				Верхня межа
Велика Омега				Нижня межа

Теорія алгоритмів. Приклад 1.

- Ціль:
  - визначити складність проблеми
  - розробити «оптимальний» алгоритм
  - Приклад: 1-Сума = «Чи є в масиві 0»?
- Верхня межа:
  - Brute-force алгоритм для 1-Суми: перебрати всі елементи масиву.
  - Час виконання O(N)
- Нижня межа: Необхідно довести, що немає алгоритму, що робить краще
  - В будь якому разі має перевірити всі N елементів (бо будь-який не перевірений елемент може бути 0)
  - Час виконання $\Omega(N)$
- Оптимальний алгоритм:
  - Верхня межа = нижній межі
  - Brute-force алгоритм для 1-Суми є оптимальним і його час виконання $\Theta(N)$

Теорія алгоритмів. Приклад 2.

- Ціль:
  - визначити складність проблеми
  - розробити «оптимальний» алгоритм
  - Приклад: 3-Сума
- Верхня межа:
  - Brute-force алгоритм для 3-Суми.
  - Час виконання $O(N^3)$
  - Але ми знайшли кращий алгоритм з складністю $O(N^2 logN)$
- Нижня межа:
  - В будь якому разі має перевірити всі N елементів
  - Час виконання $\Omega(N)$
- Відкрита проблема:
  - Необхідно знайти оптимальний алгоритм

Пам’ять

- Біт: 0 або 1
- Байт: 8 біт
- Мегабайт: 2^{20} байт
- Гігайбайт: 2^{30} байт
- Старі машини - 32-бітові машини з 4 байтовими вказівниками
- Сучасні машини - 64-бітні машини, використовують 8 байтові вказівники.
  - можуть адресувати більше пам’яті
  - вказівники займають більше пам’яті

Типові показники використання пам'яті

Тип	Байти
boolean	1
byte	1
char	2
int	4
float	4
long	8
double	8

Тип	Байти
char[]	2N+24
int[]	4N+24
double[]	8N+24

Тип	Байти
char[][]	
int[][]	
double[][]	

Типові показники використання пам'яті об'єктами в Java

- **Заголовок об'єкта**. 16 байт
- **Вказівник**. 8 байт
- **Доповнення**. Кожний об'єкт використовує розмір кратний 8 байтам, а тому інколи потрібно доповнення, що б зайняти цілий блок.
- *Приклад*. Об'єкт Date використовує 32 байти пам'яті.
  - public class Date{
    - private int day;
    - private int month;
    - private int year;
  - }

Заголовок	16 байт
day	4 байта
month	4 байта
year	4 байта
доповнення	4 байта

Приклад 2

- *Приклад*. Об'єкт String використовує ~2*N* байт пам'яті.
  - public class String{
    - private char[] value;
    - private int offset;
    - private int count;
    - private int hash;
  - }

Заголовок	16 байт
value	8+2N+24
offset	4 байта
count	4 байта
hash	4 байта
доповнення	4 байта

  - 2N+64 байт

Приклад.

- Запитання:
  - Скільки пам'яті займає об'єкт WeightedQuickUnionUF як функція від N
  - (використати нотацію ~ для відповіді)

```
public class WeightedQuickUnionUF {
    private int[] id;
    private int[] sz;
    private int count;

    public WeightedQuickUnionUF(int n){
        count = n;
        id = new int[n];
        sz = new int[n];
        for (int i = 0; i<n; i++) id[i]=i;
        for (int i = 0; i<n; i++) sz[i] = 1;
    }
    …
}
```

Приклад.

```
public class WeightedQuickUnionUF {
    private int[] id;
    private int[] sz;
    private int count;

    public WeightedQuickUnionUF(int n){
        count = n;
        id = new int[n];
        sz = new int[n];
        for (int i = 0; i<n; i++) id[i]=i;
        for (int i = 0; i<n; i++) sz[i] = 1;
    }
    …
}
```

- Відповідь:
  - Заголовок 16 байт.
  - id: 8(вказівник) +4N+24
  - sz: 8+4N+24
  - count: 4
  - Можливе доповнення 4
- 8N+88 ~8N байтів.

- Дякую за увагу.

http://seojam.ru/wp-content/uploads/2009/03/sherlock.gif