Diseño y análisis de algoritmos

Práctica 3

Grupo 11

Alberto Cabello Álvarez
Mario Valdemaro García Roque

Cuestiones sobre los ejercicios

1. Un elemento importante en el algoritmo Merkle–Hellman es la longitud de las sucesiones empleadas, lo que a su vez influye en el valor máximo de sucesión supercreciente y el módulo. Si dicha sucesión tiene N términos, intentar estimar posibles valores del término máximo y del módulo. Sugerencia: considerar el ejemplo de la sucesión Sn = 2^n , n = 0, 1, 2, . . . e intentar ampliarlo.

Siendo Sn = 2^n una sucesión superecreciente ya que se cumple que 2^n>sum(2^i) siendo i todos los índices de la sucesión hasta n-1 y por tanto el termino máximo sera 2^j donde j es el numero de datos que tenemos. En cuanto al modulo la única condición que tiene que cumplir es que sea mayor que el ultimo termino de la sucesión, es decir tiene que ser mayor que el termino máximo, por tanto cualquier numero que nos cumpla que m> 2^j nos valdrá como modulo.

Nosotros sin embargo hemos usado otro modo para generar la sucesión, consiste en ir sumando la parte anterior de la sucesión y además sumar un numero aleatorio entre 1 y 10, por tanto nos queda una expresión como esta para cada elemento de la sucesión:

Si(n)= Si(1)+Si(2)+...+Si(n-1)+rand(1,10)

Por tanto el valor máximo de la sucesión siendo esta de n elementos, sera aquella en la que todos los números generados aleatoriamente sean 10, por ejemplo para una sucesión de 5 elementos el valor máximo estaría en la siguiente sucesión:

Sn=(10, 20, 40, 80, 160)

Si nos fijamos esta sucesión tiene la siguiente expresión para generar números:

Sn(n)=10*2^n-1

Pudiendo hallar así el termino máximo de la sucesión

2. Dado el posible tamaño de los términos de la sucesión supercreciente, es necesario trabajar con enteros de tamaño adecuado. Averiguar el tamaño máximo de un entero en Python y discutir en función del mismo la longitud máxima razonable de la sucesión supercreciente.

Antes de nada vamos a enunciar algunas características que tienen Python:

A diferencia de lenguajes de programación como C Python es un lenguaje de programación no tipado, esto quiere decir que los tipos como tal no existen, es decir a la hora de crear una variable entera no la declaramos como int sino que Python hace el trabajo de ver que "tipo" es el mas adecuado en cada momento.

Comentamos esto porque existe un tipo interno en Python llamado " memory limited long " que no tiene los típicos tamaños a los que estamos acostumbrados a trabajar en C que están limitados a números de 64, 32, etc bits, sino que esta limitado a la memoria que tiene el ordenador, por lo que podemos representar números inmensos. Este tipo se activa en los casos en los que el enteros que estemos almacenando sea mayor que "92233720368547758O7" que es 2**63-1

Por tanto el número máximo de números que puede haber en esta sucesión teniendo en cuenta que este este tipo de dato pude ser casi el que nos de la gana. Si en vez de ello lo que queremos es que no se use este tipo pues tiene que se un numero menor que 2^63-1. Como el algoritmo que hemos hecho para crear esta sucesión genera números de forma aleatoria no hay un valor fijo ya que podemos tener la suerte de que ejecutemos alguna vez el algoritmo generando 61 elementos no nos pasemos y otra vez si, por tanto el número máximo de elementos que nos asegura esto es 60. Por supuesto solo si decidimos que no nos genere números por encima de 2**64 bits, sino dependerá de la memoria que tengamos.

http://stackoverflow.com/questions/5470693/python-number-limit

https://www.daniweb.com/programming/software-development/threads/71008/comparing-python-and-c-part-1-integer-variables

3. Enunciar los tiempos medio y peor esperados en el problema quickselect usando el primer elemento como pivote, así como el tiempo esperado en el caso peor cuando se usa el pivote de la "mediana de 5 elementos".

Quicksort es un algoritmo de selección cuya eficiencia depende del método de selección del ele - mento a usar como pivote dentro del algoritmo. Cada método tiene sus ventajas e inconvenientes, teniendo que hacer un trade off en algunos aspectos entre ellos.

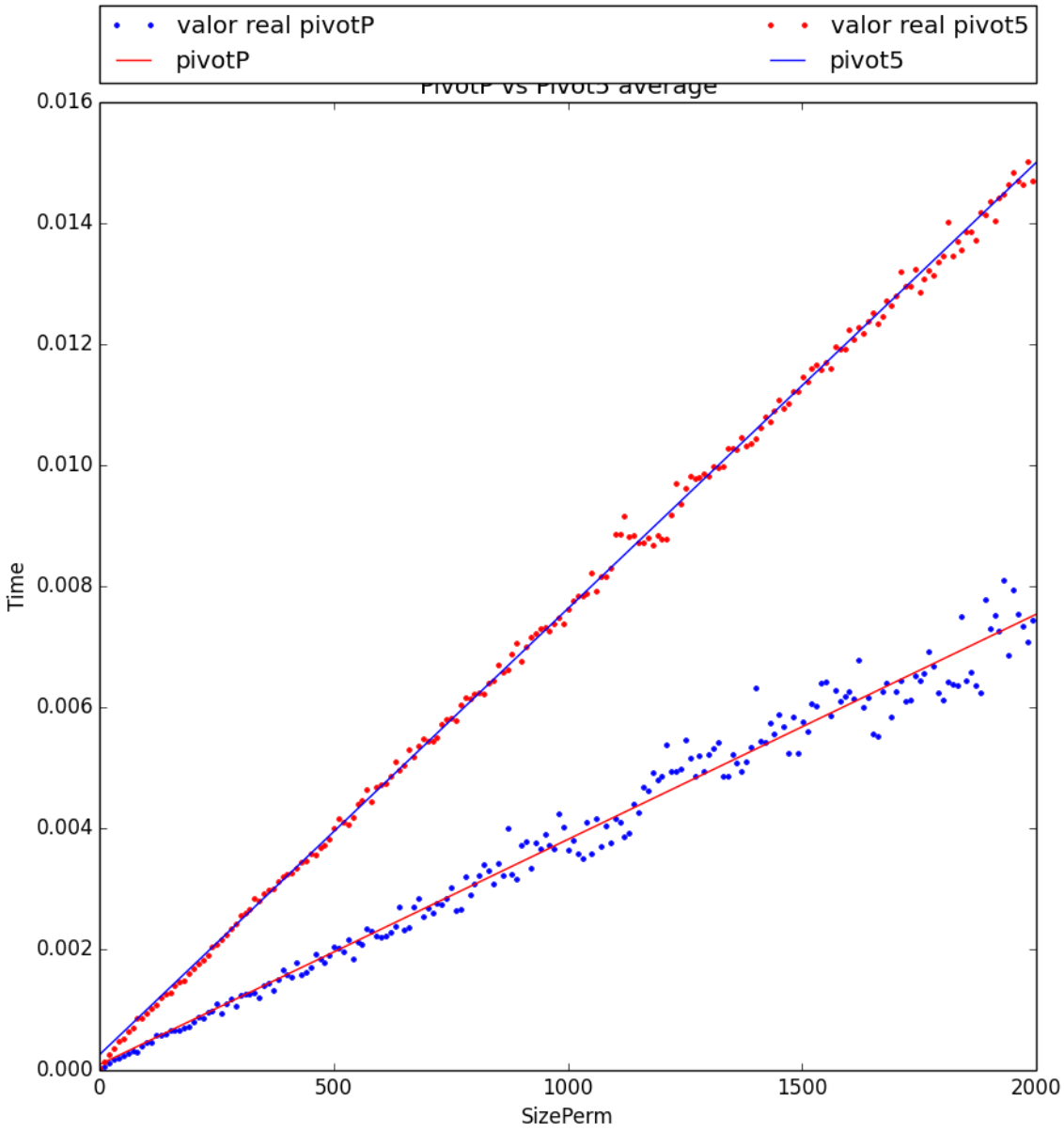
El método de seleccionar el primer elemento como pivote requiere ninguna operación adicional para esta selección, haciéndolo razonablemente rápido en el caso medio. $AQSel(N, k) \leq 4N$. Por otro lado, al igual que pasaba en quicksort, el caso peor para este método es el que la tabla está or - denada, haciendo esta elección de pivote mala, y arrojando un tiempo cuadrático. Asumiendo equi - probabilidad, esta búsqueda ocurre 1/N!, donde N es el tamaño de la tabla, aunque otras tablas razo - nablemente ordenadas también pueden resultar problemáticas en este sentido.

La técnica de mediana de 5 elementos se utiliza para evitar este efecto, para mejorar el caso peor. Se trata de un mecanismo en sí más lento, ya que tiene que hacer más operaciones para seleccionar un pivote que consiga una distribución más uniforme entre las dos mitades. El caso peor teórico para este mecanismo es de $WQS5(N) \leq 26N$

4. Utilizando las funciones desarrolladas más arriba, contrastar dichos tiempos teóricos con los valores medidos, representarlos en gráficas adecuadas y discutir los resultados que se ob - tengan.

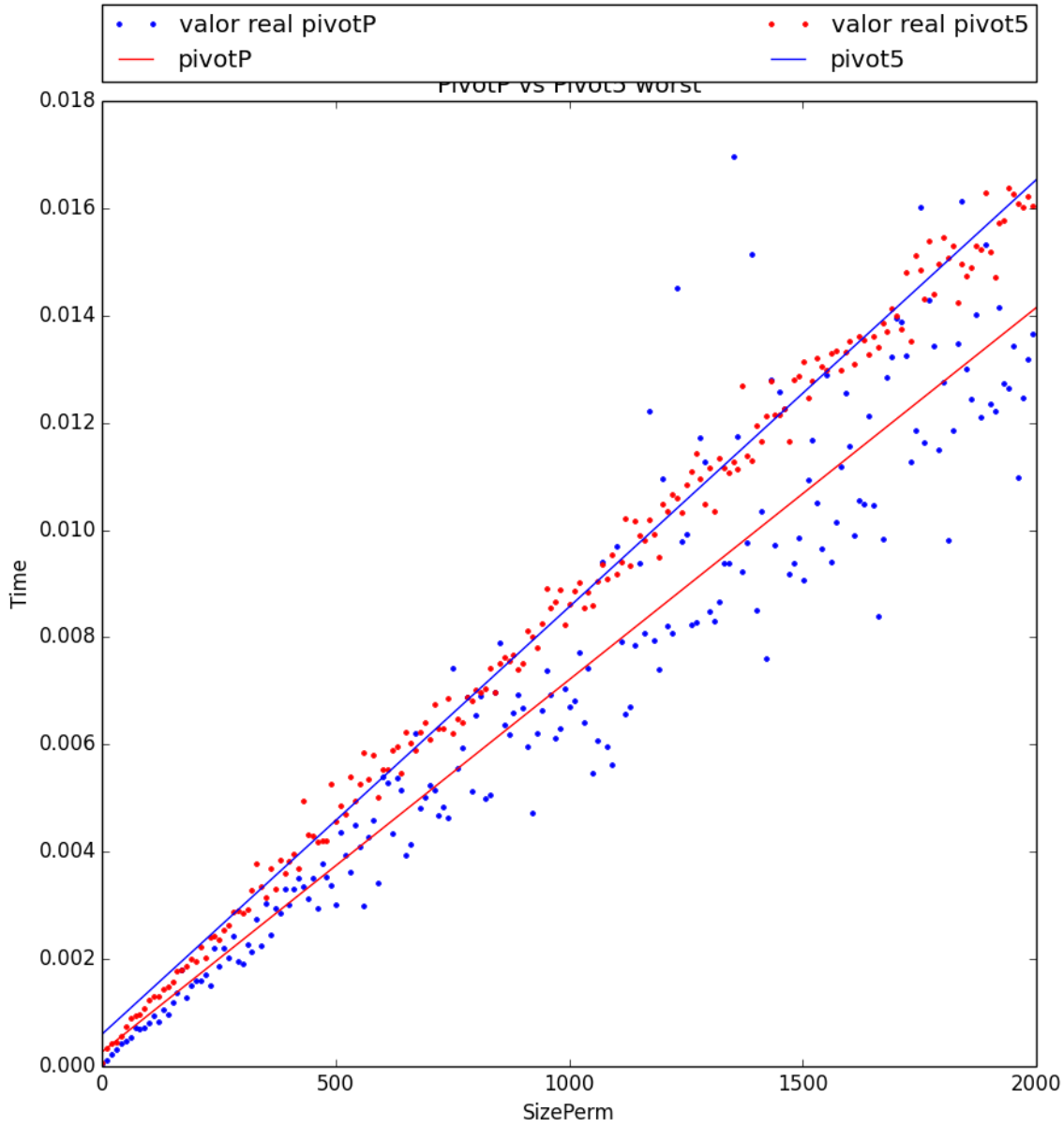
Haciendo modificaciones a la función fitPlotQselect comprobamos los casos relevantes para contrastarla eficacia de estos dos métodos.

En primer lugar, el caso medio.

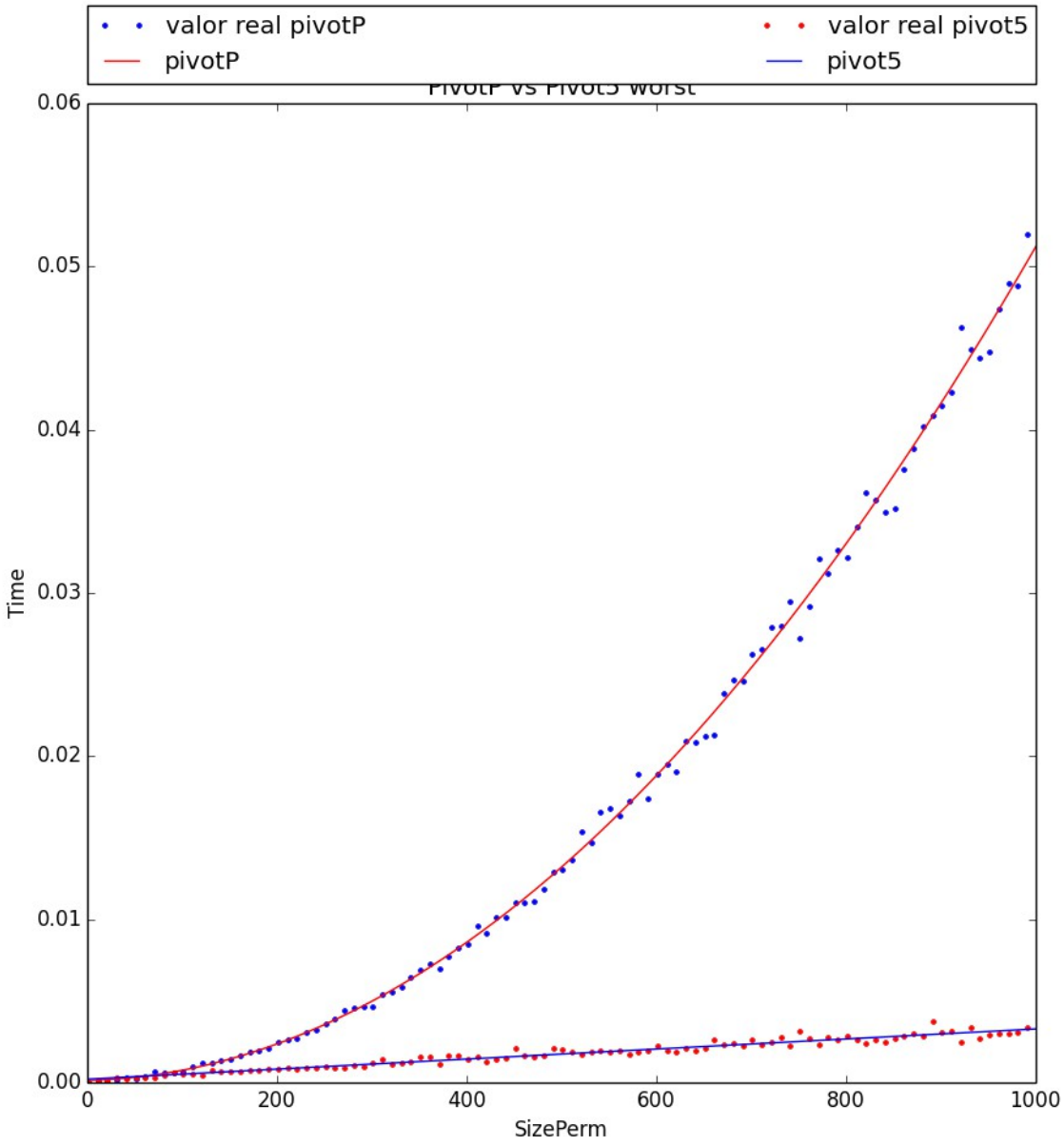


Estos resultados muestran que generalmente PivotP es más rápido que Pivot5, tan solo por la senci - llez del método que ahorra muchas operaciones en el cálculo del pivote.

Seguidamente, comparamos el caso peor, esto es, el máximo tiempo empleado en varias repeticio - nes por cada tamaño de tabla, en este caso, 50. En contraste con lo anterior, pivot5 se ve relativa - mente inafectado, resulta un método más consistente en ese sentido, no pierde su tiempo medio in - dependientemente del tipo de tabla a ordenar.



Finalmente, comprobamos el caso peor real, en el que forzamos a que la tabla en la que buscar esté ya ordenada. En este ejemplo se ve claramente el problema de pivotP, que alcanza un tiempo cuadrático para tablas ordenadas. Este es el caso límite, en el que tarda el máximo tiempo posible.



Como conclusión, según estos datos parece que lo visto en Análisis de Algoritmos sobre la elección de pivotes para Quicksort se mantiene. Un algoritmo excesivamente preciso para encontrar un pivote óptimo resulta en unos cálculos que, de media, no compensan el tiempo que tardan con el tiempo ahorrado al hacer Quickselect propiamente.

En definitiva, este problema es una cuestión de tradeoffs, usar pivote5 resulta una mejora en el caso peor, y en casos cercanos al peor, aunque en la práctica, en el caso medio, no mejora el rendimiento.