Planejamento

- Contando esta, temos 6 semanas até a P2.
  - Hoje
    - comentários sobre a prova;
    - Tabelas, Registros, Campos e Chaves e relação com Arrays, Objetos e Atributos
  - Amanhã
    - Busca (sequencial e binária);
    - Ordenação (insertion sort e bubble sort);
- EP2 no ar nesta semana.

Sugestão para resolução da prova

Uma tarefa (lista/prova bem confeccionada)

- Apresenta um problema “inédito” que pode ser resolvido com os conceitos apresentados em aula.
- Testa direta ou indiretamente se os conceitos importantes (nome e definição) foram bem entendidos.
- Testa a capacidade de expressão (vocabulário geral e específico) e organização (estruturação/encadeamento de idéias)
- Testa a estratégia de resolução.

Uma tarefa (lista/prova bem resolvida)

- É feita para facilitar/agilizar o entendimento de quem lê/corrige.
- É aderente a uma estratégia de resolução.
- Tem respostas no formato adequado
  - dissertativa (Diga o que vc entende por... Justifique...)
    - mostra que entende
  - pontuada, organizada em ítens (Cite N, apresente N...)
    - até dá para resolver só com decoreba
  - tabelas (Cite N, apresente N...)
    - até dá para resolver só com decoreba
  - fórmulas, programas (Resolva...)
    - há quem prefira avaliar apenas a resposta – fácil de colar;
    - há quem prefira avaliar o que foi feito – menos fácil de colar.
  - diagramas/esquemas/desenhos (...)
    - um bom diagrama facilita o entendimento e demonstra organização de idéias

Qual a dificuldade?

- Organizar/classificar/encadear conceitos e idéias de forma que eles sejam prontamente usados.
- Quem decora conceitos/fórmulas mas não sabe aplicá-los em coisas do dia-a-dia não se dá bem com esse tipo de avaliação.
- Não há uma única maneira de resolver um problema.
- As possibilidades (tanto de problemas quanto de soluções) são infinitas.
- Existe um viés em nossa formação: somos pouco estimulados a observar, analisar e questionar racionalmente e muito estimulados a decorar.

Somos pouco estimulados a observar, analisar e questionar racionalmente

- Porque é mais fácil ter N maquininhas que fazem tudo igual, limitadas e bem comportadas (quietinhas, com cara de nuvem, eventualmente não entendendo nada).
- As pessoas são tratadas de forma a se tornar conformadas, dóceis e mal resolvidas.
- O povo é pacífico, incapaz de argumentar e evoluir – é facilmente controlável.
- Citando Feynman, 1951 – brasileiro decora os conceitos/fórmulas, mas não é capaz de vê-los aplicados a processos do dia-a-dia.
- Num certo sentido é uma deficiência com que temos que lidar agora (deveria se fazer isso desde o fundamental)

Nosso desafio é superar este estágio.

O que é preciso?

- Esforço/dedicação
- Maturidade
- Questionar sem transformar a discussão em questão pessoal
- Inteligência, flexibilidade (como vou expicar o que quero??)
- Metodologia???
- Aprender ainda é um processo individual, cabe a cada um encontrar o seu processo.

O que vamos sentir?

- Frustração
- Raiva
- Pena
- Tristeza
- Alegria
- Orgulho
- Satisfação
- ...

Existe relação entre esta matéria e a da aula anterior?? Qual é? Faz sentido combinar o tópico A com o tópico B? Como faço?

São perguntas fundamentais e permitidas em aula – respondo na medida do possível.

Agora vamos à prova!

5 minutos iniciais: ler todas as questões e detalhar a estratégia

- Estratégia geral: Estimar o benefício de se responder a questão: Ponderar tempo estimado para solução e valor da questão. Respoder a partir da questão que traz maior benefício.

- Estimar o tempo a gastar para cada questão.
- Estimar a nota – supondo que se acerta uma parte da questão (0..100%), atribuir as notas para cada questão.
- Com uma estimativa conservadora de tempo e nota, qual a nota final estimada? É satisfatória?
  - Sim: então “tá dominado!”
  - Não: o que fazer?? Sugestão: resolver mais rápido e usar o tempo restante para salvar uns “centavos”

Q1

- Testar conceitos sobre paradigmas de linguagens, linguagens orientadas a objeto e Java
- Testar capacidade de expressar/concatenar idéias.

Q1 – slide da aula 7

Paradigma	Descrição	Característica notável / exemplo
Imperativo	Computação em termos de comandos que diretamente mudam o estado do programa	Atribuição direta, variáveis globais
Estruturado	Paradigma derivado do Imperativo em que a estrutura é mais facilmente entendida	Indentação, ausência de GOTO
Funcional	Trata computação como a avaliação de funções matemáticas. Não é possível mudar o estado fora da função.	Lambda-cálculo, Scheme, Haskell
Procedural	Derivado do paradigma estruturado e da noção de código modular (subrotinas e funções com efeitos colaterais)	Variáveis locais, procedimentos, modularização
Orientado a eventos	O fluxo do programa é determinado por eventos, como apertar botões no mouse…	Eventos, Loop de eventos, apontadores de eventos, execução assíncrona
Orientado a objetos	Trata conjuntos de variáveis e procedimentos (métodos) como objetos.	Classes, objetos, atributos, métodos, assinatura, passagem de mensagens, instanciação, herança, polimorfismo, sobrecarga, sobrescrita.

OOP

- Retenção de estado e relação com referências
- Classe, objeto, instância, método, atributo
- Escopo e modificadores de acesso
- Ocultamento, encapsulamento e modificadores de acesso

Conceitos

- **Objeto** (em OOP) é uma abstração de uma “coisa” (para diferenciar de objeto concreto, abstrato,…em linguagem natural)
- **Abstração** é o processo pelo qual retemos o que é essencial para um dado objetivo, eliminando características irrelevantes
- **Classe** é um template para um objeto. Ela contém a definição dos atributos de um objeto (OOP) e os métodos que o objeto pode executar.
- **Atributos** são as características ou propriedades de um objeto e sempre são variáveis da classe.
- **Métodos** são um conjunto de procedimentos que pode ser invocados (chamados) para que se execute algo com o objeto.
- **Instância** é um membro de uma classe, que depois de corretamente preenchido, em tempo de execução, corresponde a um objeto.
- **Mensagem** é o nome dado ao processo de envio de dados de um objeto para outro. Em JAVA a mensagem para um objeto consiste apenas na invocação de um método desse objeto.
- **Encapsular** consiste em limitar escopos e construir códigos com módulos bem definidos.
- **Herança** é o nome dado à capacidade de uma classe ter os mesmos atributos e métodos de outra sem ser necessário duplicar código.
- **Polimorfismo** é o nome dado à possibilidade de existência de vários métodos de mesmo nome com comportamentos diferentes.

Tipos Abstratos de Dados

- Ocultamento da informação e implementação
  - Permite que possa alterar a implementação do tipo (contanto que mantenha as mesmas operações) sem afetar as unidades de programa que fazem uso dele
  - Aumenta a confiabilidade, pois nenhuma outra unidade de programa pode mudar, acidentalmente ou intencionalmente, as representações do tipo, aumentando a integridade de tais objetos

Orientação a Objetos

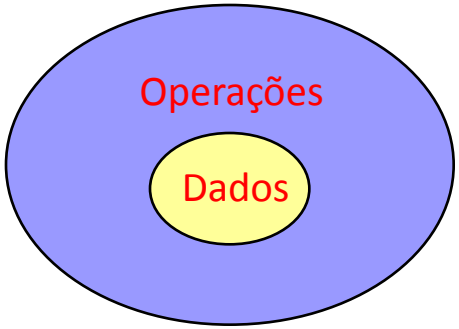
- Pode ser visto como uma extensão do conceito de Tipos Abstratos de Dados
  - Combinação de dados e operações (sobre eles) em um elemento único

- Classe: definição do Tipo Abstrato de Dados
- Objeto: cada instância derivada da classe

- Representa em software entidades que encontramos no mundo real

Encapsulamento

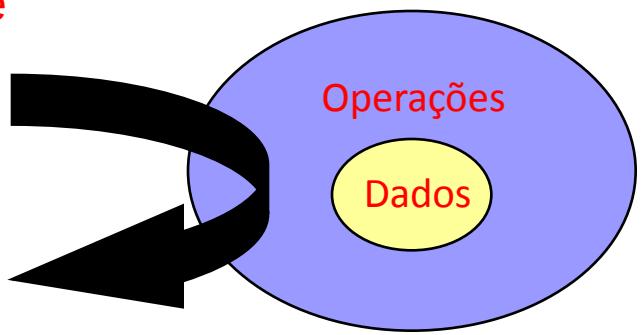
- Paradigmas convencionais separam dados e procedimentos
- O objeto contém tantos os dados quanto as operações:
  - Dados: atributos
  - Implementação das operações: métodos



Encapsulamento e Ocultamento de Informações

- Utilização de encapsulamento para restringir a visibilidade externa de certos detalhes de informações (dados) ou implementações (operações), os quais são internos à estrutura de encapsulamento

- Não é possível chegar as dados diretamente
- Cliente não tem conhecimento acerca de como as operações são implementadas



Encapsulamento e Ocultamento de Informações

- **Encapsulamento**: Dados e os Métodos são "empacotados" em um objeto, de forma que objetos externos somente tenham acesso àquilo que for permitido pelo objeto
- O encapsulamento facilita o ocultamento de informações, separando aspectos externos de um objeto, que são acessíveis para outros objetos, dos detalhes internos de estrutura (dados) e implementação, que ficam "escondidos" de outros objetos
  - restringe a visibilidade do objeto, mas facilita reuso
  - atributos e métodos empacotados sob um só nome e podem ser reusados como uma especificação ou componente de programa

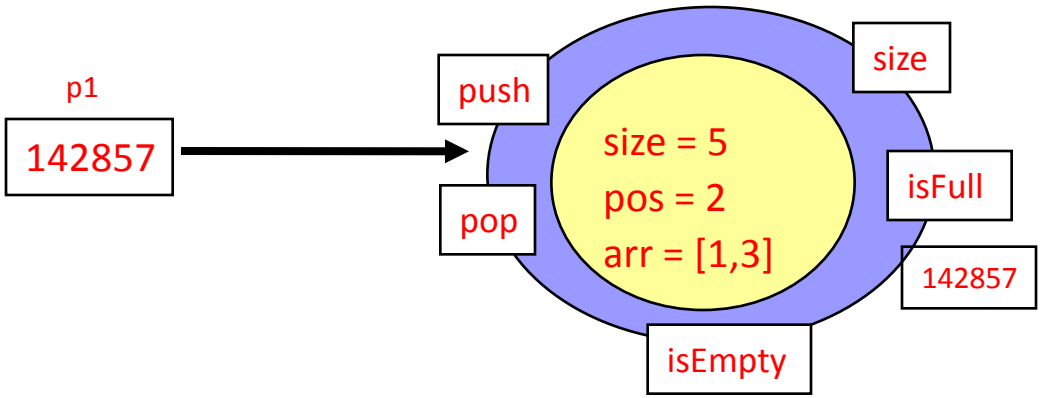
Retenção de Estado

- Habilidade de um objeto reter seu estado
  - estado ≅ conjunto de valores de seus atributos

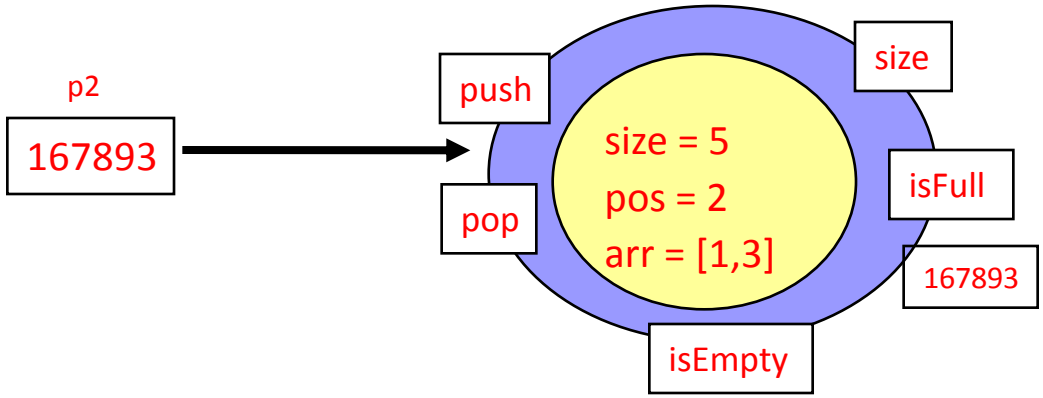
- Um objeto é ciente de seu passado (operações que foram executadas previamente)
- O estado influencia o comportamento do objeto
  - método pop() retorna o último item inserido através do método push()
  - método size() retorna valores diferentes antes e após a execução de um push() ou pop()

Referência para objetos

```
Pilha p1 = new Pilha(5);
p1.push(1);
p1.push(3);
```




```
Pilha p2 = new Pilha(5);
p2.push(1);
p2.push(3);
```



Modificadores de acesso 07.04

- public
- protected
- package protected
- private

- http://download.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

Access Levels				
Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Q2

- Verificar habilidade em substituir construções de linguagens de programação mantendo a função
- Verificar conhecimento do teorema da programação estruturada
- Verificar noção prática do processo de compilação (esta é uma maneira de resolver)

Q2

```
int Dezena, Unidade, Decremento=2, DezenaInicial=7, DezenaFinal=0,
    UnidadeInicial=6, UnidadeFinal=0;
Dezena=DezenaInicial;
Unidade=UnidadeInicial;
do {
      do {
            System.out.print (Dezena);
            System.out.print (Unidade);
            System.out.println ("");
            Unidade-=Decremento;
      } while (Unidade!=UnidadeFinal);
      Unidade=UnidadeInicial;
      Dezena--;
} while (Dezena!=DezenaFinal);
```

End	Mem	Mne	descrição
0	+1132	entry:CEA DezenaInicial	
1	+1230	CAE Dezena	Dezena=DezenaInicial;
2	+1133	CEA UnidadeInicial	
3	+1131	CAE Unidade	Unidade=UnidadeInicial;
4	+5000	do: NOP	do { do {
5	+1130	CEA Dezena	
6	+5800	IDI	System.out.println (Dezena)
7	+1131	CEA Unidade	
8	+5800	IDI	System.out.println (Unidade)
9	+2236	SUB Decremento	
10	+1131	CAE Unidade	Unidade-=Decremento
11	+2235	SUB UnidadeFinal	
12	+5304	DPZ do:	} while (Unidade>=UnidadeFinal)
13	+1133	CEA UnidadeInicial	
14	+1131	CAE Unidade	Unidade=UnidadeInicial
15	+1130	CEA Dezena	
16	+2237	SUB DecrementoDezena	Dezena--;
17	+1230	CAE Dezena	
18	+1132	SUB DezenaFinal	
19	+5304	DPZ do:	} while (Dezena>=DezenaFinal)
20	+7000	PAR	Fim

End	Mem	Mne
30		Dezena
31		Unidade
32	+0007	DezenaInicial
33	+0006	UnidadeInicial
34	+0000	DezenaFinal
35	+0000	UnidadeFinal
36	+0002	Decremento
37	+0001	DecrementoDezena

Slide da aula 4

Endereço	Conteúdo	Linguagem de montagem	Explicação por extenso
1	+1130	CEA zero	Copie o conteúdo do endereço 30 no acumulador
2	+1240	CAE soma	Copie o conteúdo do acumulador no endereço 40
3	+3150	leia: LER num	Leia um número e coloque no endereço 50
4	+4150	IMP num	Imprima o conteúdo do endereço 50
5	+1150	CEA num	Copie o conteúdo do endereço 50 no acumulador
6	+5411	DNE fim	Se o conteúdo do acumulador for menor que zero, desvie para o endereço 11
7	+1140	CEA soma	Copie o conteúdo do endereço 40 no acumulador
8	+2150	SOM num	Some o conteúdo do endereço 50 com o conteúdo do acumulador e guarde no acumulador
9	+1240	CAE soma	Copie o conteúdo do acumulador no endereço 40
10	+5103	DES leia	Desvie para o endereço 03
11	+4140	fim: IMP soma	Imprima o conteúdo do endereço 40
12	+7000	PAR	Pare
30	0	zero:	Variável com valor zero
40		soma:	Variável de nome "soma"
50		num:	Variável de nome "num"

Slide da aula 4

```
            zero = 0;
            acumulador = 0;
            soma = 0;
LEIA:       num = leia ();
            imprima (num);
            acumulador=num;
            if (acumulador<0) {
                        imprima ();
                        fim ();
            }
            else {
                        acumulador=soma;
                        acumulador=acumulador+num;
                        soma=acumulador;
                        salte para LEIA;
            }
```

- ... mais na semana que vem.

- NOTA: mais para a frente no curso veremos que este código foi BEM mal escrito !! Mas é exatamente o que está no programa do HIPO.

Q3

- Verificar se sabe o que é base numérica
- Verificar se sabe converter bases rapidamente
- Verificar se sabe o que é representação em complemento de dois
- Verificar se sabe converter para representação em complemento de dois.

Q3

- $(25)_8 = (010101)_2 = (15)_{16} = (1*16+5*16^0)_{10} = (21)_{10}$

cada digito octal corresponde a 3 bits

agrupa de 4 em 4 bits

converte para decimal

- $(59)_{10} = (111011)_2 = (3B)_{16} = (73)_8$

divisões sucessivas

agrupa de 4 em 4 bits

agrupa de 3 em 3 bits

- Complementa e soma 1
- $\sim(010101)_2+1=(101010)_2+1=(101011)_2$
- $\sim(111011)_2+1=(000100)_2+1=(000101)_2$

Q4

- Verificar se lembra o que fez no EP1.
- Verificar se sabe o que é static.
- O problema do troco e o do caixa eletrônico não são parecidos? Como transformar um no outro??

```
class Caixa {

   /** Valores de face de cada nota */
   public static final int FaceA=4096, FaceB=512, FaceC=64,
                           FaceD=8, FaceE=1;

   /** Quantidades de notas */
   public static int NotasA, NotasB, NotasC, NotasD, NotasE;

   /** Imprime na tela a quantidade de notas de cada valor que
    * devem ser devolvidas.
    */
   public void imprimeNotas () {
      System.out.println ("Notas de " + FaceA + ":" + NotasA);
      System.out.println ("Notas de " + FaceB + ":" + NotasB);
      System.out.println ("Notas de " + FaceC + ":" + NotasC);
      System.out.println ("Notas de " + FaceD + ":" + NotasD);
      System.out.println ("Notas de " + FaceE + ":" + NotasE);
   }

public static void main (String args[]) {
      Caixa Cx1, Cx2, Cx3;

      Cx1=new Caixa();
      Cx2=new Caixa();

      Cx3=Cx1;
      Cx3.calculaTroco (2777, 1, 0, 0, 0, 0);
      Cx2.imprimeNotas ();
   }
}
```

```
/** Recebe o Valor da mercadoria que o cliente comprou e a
      quantidade
    * de notas de cada valor de face que o cliente entregou. Calcula
    * o troco e armazena em A, B, C, D e E, a quantidade mínima de notas
    * que tem que ser devolvidas ao cliente.
    */

   public void calculaTroco  (int Valor, int NotasA, int NotasB,
                       int NotasC, int NotasD, int NotasE) {
      int Troco=(NotasA*FaceA+NotasB*FaceB+NotasC*FaceC+
                  NotasD*FaceD+NotasE*FaceE) - Valor;
      System.out.println ("Troco=" + Troco);
      if (Troco<0) {
         System.out.println ("Caloteiro!");
      }
      else {
         this.NotasA=Troco/FaceA;
         Troco%=FaceA;
         this.NotasB=Troco/FaceB;
         Troco%=FaceB;
         this.NotasC=Troco/FaceC;
         Troco%=FaceC;
         this.NotasD=Troco/FaceD;
         Troco%=FaceD;
         this.NotasE=Troco;
      }
   }
---------- Resultado da execução (tanto para Q4 quanto para Q5)
F:\ICC1\prova>java Caixa
Troco=1319
Notas de 4096:0
Notas de 512:2
Notas de 64:4
Notas de 8:4
Notas de 1:7
```

Q5

1. -
2. declara referências
3. -
4. instancia Cx1
5. instancia Cx2
6. -
7. Cx3 e Cx1 se referenciam ao mm objeto
8. chama método de Cx3
9. chama método de Cx2

a) quadro anterior

b) compila

c) Troco=1319

Notas de 4096:0

Notas de 512:2

Notas de 64:4

Notas de 8:4

Notas de 1:7

d) Não pois Cx3 e Cx1 se referem ao mm objeto. Mesmo que isso fosse corrigido, os atributos da classe são static, logo são compartilhados por todas as instâncias da classe.

A prova não estava difícil

Fáceis (leva pouco tempo)

- Q3 é aplicação de fórmulas (1pt)
- Q1 é memorização de conceitos (2pt)
- Q4 é igual ao EP1 (2,5pt)

Difíceis (leva mais tempo)

- Q5 precisa ler o fonte, saber como funciona cada comando/modificador e ligar as idéias (2,5pt)
- Q2 é trabalhosa. Fazer a tradução de Java para LM é "mecânico", mas requer uma "sacada". A abordagem direta é a mais difícil para quem não tem experiência (2pt)

Por que estudar busca e ordenação

- Estes algoritmos são muito utilizados;
  - Sistemas operacionais;
  - Bancos de dados;
  - ...
- Induzem a criação de várias estruturas de dados (filas de prioridade, heap, hashtable, árvores B,...) (AED1 e 2);
- São fáceis de entender (este é nosso objetivo agora);
- São usados para introduzir técnicas de análise de algoritmos (ICC2).

Referências, objetos, registros, chaves,...

- Classe Aluno
  - Nome
  - numero
  - notas[]
  - foto
- Turma é um array de alunos.
  - um elemento de Turma contém uma instância do objeto aluno??
  - Isto está ligado ao conceito de índices em bancos de dados.
- Escreva um método que recebe um número e busca o aluno com esse número, imprimindo seu nome.
- Neste contexto, Turma é uma tabela que contém registros (instâncias). Chaves de busca são os campos (atributos) que testamos para fazer buscas.

Os números poderiam ser armazenados diretamente no array Turma? Apresente o código adaptado.

Note que o código adaptado mantém a estrutura do código inicial

- ... então, por simplicidade, pode-se considerar arrays de inteiros (chaves) equivalentes a arrays de registros.
- Isto nos poupa do trabalho de lidar com códigos extensos e de preocupações ligadas a utilidade e não ao desempenho.

Algoritmos de Busca

- Busca linear
  - acabamos de fazer – varrer o array linearmente.
- Busca binária
  - requer que o array esteja ordenado (tenha uma certa estrutura.
  - Em geral, quando qualifica-se um campo em um banco de dados como chave, o gerenciador de banco de dados mantém a tabela ordenada pelo campo, ou mantém um array de referências aos registros onde a relação de ordem está codificada de alguma forma.

Algoritmos de ordenação (sorting)

- Insertion sort
- Bubble sort
- merge, heap, quick – algoritmos “eficientes”.

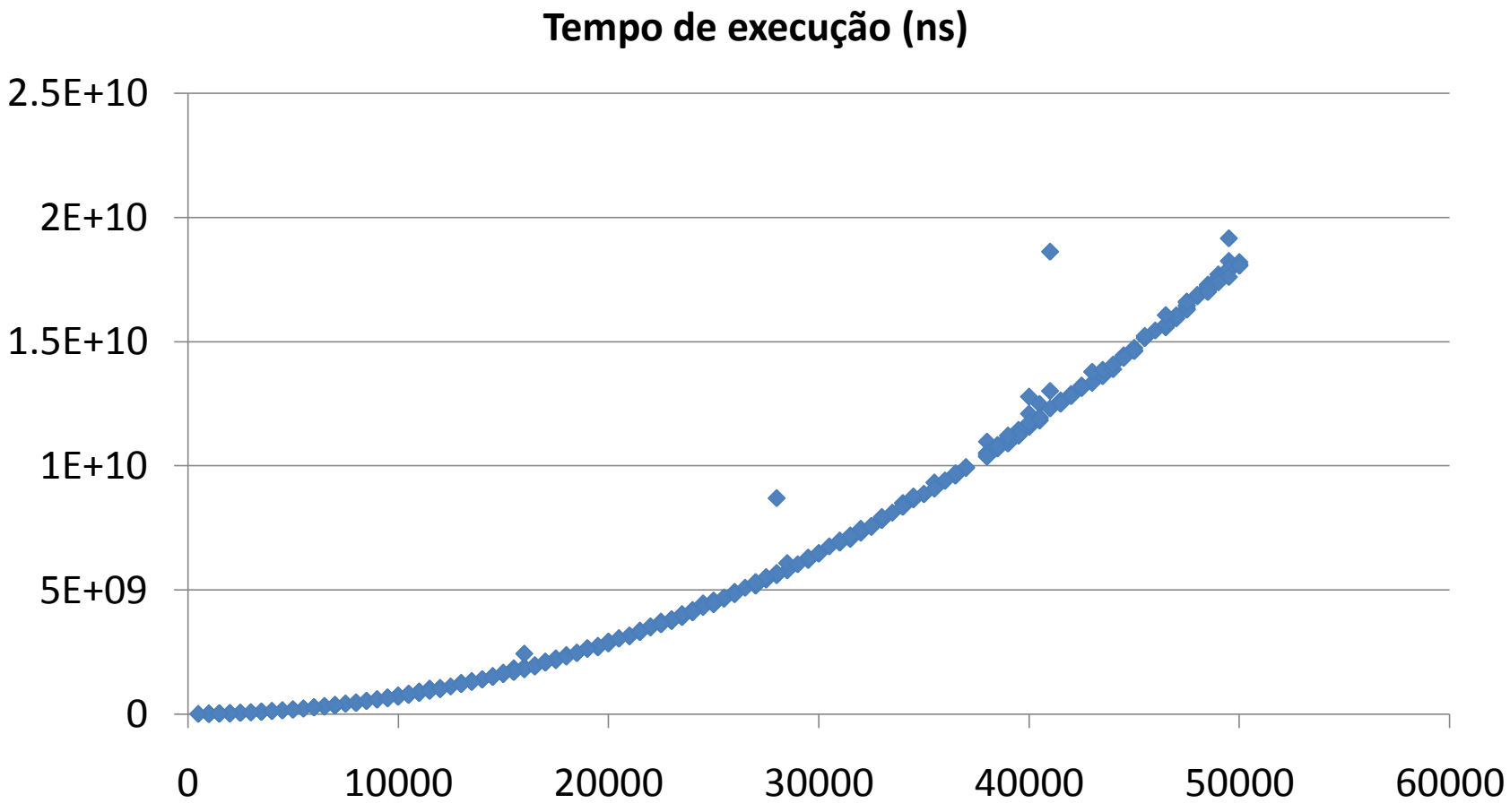
Uma espiada em ICC2

- O que queremos dos nossos programas?
  - Executar eficientemente
  - Usar bem os recursos da máquina
  - Resolver em tempo razoável
- Para isso podemos medir diretamente a memória alocada e o tempo de execução, mas esses valores flutuam e são difíceis de medir.
- A simples medição não é muito ilustrativa para apontar possíveis melhorias.
- Por isso, analisamos algoritmos.

Medidas diretas

- Tempo de execução:
  - StartTime=System.nanoTime(); /* java.util.* */
  - EndTime=System.nanoTime();
- Memória utilizada:
  - Contar quantas instâncias existem (como??)
  - Java não garante que Finalize() seja chamado imediatamente quando deixamos de usar um objeto. Mas em C++ isso é possível

Bubble


Tempo de execução (ns)
2.5E+10
2E+10
1.5E+10
1E+10
5E+09
0
0
10000
20000
30000
40000
50000
60000